Prediction without Preclusion: Recourse Verification with Reachable Sets

Avni Kothari[1], Bogdan Kulynych[2], Tsui-Wei Weng[1], and Berk Ustun[1]

[1]UC San Diego [2]EPFL

Abstract

Machine learning models are often used to decide who will receive a loan, a job interview, or a public benefit. Standard techniques to build these models use features about people but overlook their *actionability*. In turn, models can assign predictions that are *fixed* – meaning that consumers who are denied loans, interviews, or benefits may be permanently locked out from access to credit, employment, or healthcare. In this work, we introduce a formal testing procedure to flag models that assign fixed predictions that we call *recourse verification*. We develop machinery to reliably determine if a given model can provide recourse to its decision subjects from a set of user-specified actionability constraints. We demonstrate how our tools can ensure recourse and adversarial robustness in real-world datasets and use them to study the infeasibility of recourse in real-world lending datasets. Our results highlight how models can inadvertently assign fixed predictions that permanently bar access, and we provide tools to design algorithms that account for actionability when developing models.

1 Introduction

Machine learning models routinely assign predictions to *people* – be it to approve an applicant for a loan [34], a job interview [8, 65], or a public benefit [83, 20, 24]. Models in such applications use features that capture individual characteristics without accounting for how individuals can change them. In turn, models may assign predictions that are not responsive to the *actions* of their decision subjects. In effect, even the most accurate model can assign a prediction that is *fixed* (see Fig. 1). The responsiveness of machine learning models to actions of decision subjects is vital to safety of the models in consumer-facing applications. In fraud detection and content moderation [36, 55, 45], for example, models should assign predictions that are fixed to prevent malicious actors from circumventing detection. In lending and hiring, however, these predictions should exhibit *some* sensitivity to actions. Otherwise, models that deny loans and interviews could *preclude access* to credit and employment, thus violating rights and regulations such as equal opportunity [4] and universal access [14].

There is a broad lack of awareness that models in lending and hiring could inadvertently assign fixed predictions. In other words, models might not provide *recourse* [73, 75, 41], i.e., the ability of data subjects to change their predictions through feasible actions. In this work, we propose to test this effect by certifying the feasibility or infeasibility of recourse. In contrast to prior work on actionable recourse [42, 76], which aims to find actions that provide recourse, our goal is *verification* of whether such actions exist in the first place. Verification is a challenging computational task on its own. We show that if one were to use a standard method for recourse provision, such procedure could either return "incorrect" actions that are not feasible, or fail to find an action where one exists.

Features		Actionability Constraints	Reachable Set	Label Freq.		Best Model	
`prior_applicant` (x_1)	`age`$\geq$`60` (x_2)	$A(x_1, x_2)$	$R_A(x_1, x_2)$	n^-	n^+	$\hat{f}$	$\hat{R}(\hat{f})$
1	1		$\{(1,1)\}$	27	15	-	15
0	1	$a_1 \geq 0,\ a_1 + x_1 \in \{0,1\}$	$\{(1,1),(0,1)\}$	11	25	+	11
1	0	$a_2 \geq 0,\ a_2 + x_2 \in \{0,1\}$	$\{(1,1),(1,0)\}$	12	25	+	12
0	0		$\{(1,1),(0,1),(1,0),(0,0)\}$	10	25	+	10

Figure 1: A toy example of a classification task where the most accurate linear classifier assigns a prediction without recourse to a fixed point (highlighted in red). We predict $y =$ `repay_loan` using two features $(x_1, x_2) =$ (`prior_applicant`, `age`$\geq$`60`) that can only change from 0 to 1. Feasible actions that individuals can perform are denoted as a_1 and a_2, applied as $x_1 + a_1$ and $x_2 + a_2$ to obtain a new *reachable* point $(x_1 + a_1,\ x_2 + a_2)$. In this example, the point $(1, 1)$ (thus, `prior_applicant`, `age`$\geq$`60`) is a *fixed point* that will be assigned a *prediction without recourse* by any classifier f such that $f(1, 1) = -1$. We show that this point is assigned a prediction without recourse by $\hat{f}$, the optimal empirical risk minimizer for a dataset with $n^- = 60$ negative examples and $n^+ = 90$ positive examples.

We propose a model-agnostic approach for the task of recourse verification. Our method solves the problem by generating *reachable sets*, i.e., regions of feature space that are confined by actionability constraints, and querying models on reachable sets in a black-box way. Reachable sets are a powerful tool for verification because they can be constructed directly from the actionability constraints, and can be used to characterize the feasibility of recourse for *any model*. In particular, any model will assign a prediction without recourse if it fails to assign a target prediction over its reachable set. The main contributions of our work include:

1. We introduce a model-agnostic approach for recourse verification with reachable sets. We identify salient classes of reachable sets and develop theory to show how they can be used to design practical verification algorithms.

2. We develop fast algorithms to delineate reachable sets from complex actionability constraints. Our algorithms can be used to ensure that a model can provide recourse in model development or deployment, and are designed to abstain when they are unable to certify recourse in order to avoid incorrect outputs.

3. We present an empirical study of the infeasibility of recourse using several real-world datasets, realistic actionability constraints, and common model classes. Our results illustrate the prevalence of predictions without recourse in lending applications, and highlight pitfalls in flagging these examples with recourse provision. Finally, we demonstrate how our methods can be used to ensure recourse in consumer-facing applications like lending and content moderation.

4. We provide a Python library that implements our tools, available in this anonymized repository.

Related Work This work opens a new direction for research on *algorithmic recourse*, which studies how to change the prediction of a given model through *actions* in a feature space [73, 75]. Much work on recourse develops methods for *recourse provision* – i.e., methods to provide a person with an action to change the prediction of a given model or, relatedly, *counterfactual explanations* – i.e., methods that explain a model's decision by showing what actions would change it [see e.g., 35, 18, 59, 42, 76, 77, 66, 38]. We focus instead on *verification* of models in terms of recourse feasibility – i.e., testing if a model assigns predictions that a given person can change using any feasible action. The need for verification arises because algorithmic recourse may be infeasible under realistic actionability constraints. Although actionability is a defining characteristic of recourse [see e.g., 75], the fact that such constraints may lead to infeasibility is not well-known in the literature. The exceptions [73, 43, 16] mention infeasibility but do not study it in detail. In contrast to the

lack of attention in the literature, we show that recourse infeasibility is pervasive and is completely missed by most of the existing methods for recourse provision.

Our work is related to a stream of methods that solve combinatorial optimization problems to find a recourse action or counterfactual explanations [73, 71, 61]. These methods solve combinatorial optimization problems that embed a model and discrete constraints. We solve simpler variants of this problem that only embed deterministic actionability constraints to enumerate sets that can be used for model-agnostic verification. Our approach can flag infeasibility for models that would be hard or impossible to embed in a combinatorial optimization problem (e.g., random forests, ensembles, deep neural networks).

Our work primarily involves verification as a procedure to protect access in consumer-facing applications. This task is essential to ensure the viability of recourse provision, as there are no actions to provide when recourse is infeasible. The latter motivation is shared by a stream of work on the robustness of recourse with respect to distributions shifts [67, 26], model updates [72, 62], group dynamics [3, 60, 29], and causal effects [53, 41, 78]. More broadly, testing recourse infeasibility is valuable for eliciting individual preferences over recourse actions [80, 85, 84], measuring the effort required to obtain a target prediction [32, 27], building classifiers that incentivize improvement [70, 46, 47, 2, 31], or preventing strategic manipulation [28, 17, 51, 57, 56, 22, 10, 30]. Our tools can support other verification tasks that test the sensitivity of model predictions to semantically meaningful changes to inputs, such as ensuring adversarial robustness in tabular tasks [52, 39, 33, 79, 55, 45] or stress testing for counterfactual invariance [74, 54, 64].

2 Recourse Verification

Basics We consider a standard classification task where we are given a model $f : \mathcal{X} \to \mathcal{Y}$ to predict a label $y \in \mathcal{Y} = \{-1, +1\}$ using a *feature vector* $\boldsymbol{x} = [x_1, \ldots, x_d]$ in a bounded feature space $\mathcal{X}_1 \times \ldots \times \mathcal{X}_d = \mathcal{X}$. We assume each instance represents a person and that $f(\boldsymbol{x}) = +1$ represents a desired *target prediction* (e.g., applicant repays their loan within 2 years). In the rest of the paper, we use terms *feature vector* and *point* in a feature space interchangeably.

We study if a person can attain a target prediction from a model by performing *actions* on their features. We represent each *action* as a vector $\boldsymbol{a} = [a_1, \ldots, a_d] \in \mathbb{R}^d$ that would shift their features from $\boldsymbol{x}$ to $\boldsymbol{x} + \boldsymbol{a} = \boldsymbol{x}' \in \mathcal{X}$. We denote the set of all actions available to a person with a feature vector $\boldsymbol{x} \in \mathcal{X}$ through the *action set* $A(\boldsymbol{x})$, and assume that $\mathbf{0} \in A(\boldsymbol{x})$. We do not make any assumptions on the action set other than it being deterministic, and dependent on $\boldsymbol{x}$. In practice, action sets are compactly encoded through sets of constraints on $a_1, \ldots, a_d$ (see Fig. 1 for an illustration).

Recourse Provision Given a model $f : \mathcal{X} \to \mathcal{Y}$ and a point $\boldsymbol{x} \in \mathcal{X}$ with action set $\boldsymbol{a} \in A(\boldsymbol{x})$, the *recourse provision* task seeks to find an action to attain a target prediction by solving an optimization problem of the form:

$$\begin{aligned} \min_{\boldsymbol{a}} \quad & \text{cost}(\boldsymbol{a} \mid \boldsymbol{x}) \\ \text{s.t.} \quad & f(\boldsymbol{x} + \boldsymbol{a}) = +1, \\ & \boldsymbol{a} \in A(\boldsymbol{x}), \end{aligned} \tag{1}$$

where $\text{cost}(\boldsymbol{a} \mid \boldsymbol{x}) \geq 0$ is a *cost function* that captures the cost of enacting $\boldsymbol{a}$ from $\boldsymbol{x}$ [73]. This is a computationally challenging optimization problem because it combines two classes of difficult constraints:

Condition	Sep.	Cvx.	Example	Features	Constraint
Immutability	✓	✓	`n_dependents` should not change	$x_j =$ `n_dependents`	$a_j = 0$
Monotonicity	✓	✓	`prior_applicant` can only increase	$x_j =$ `prior_applicant`	$a_j \geq 0$
Integrality	✓	✗	`n_accounts` must be positive integer ≤ 10	$x_j =$ `n_accounts`	$a_j \in \mathbb{Z} \cap [0 - x_j, 10 - x_j]$
Categorical Feature Encoding	✗	✗	preserve one-hot encoding of `married`, `single`	$x_j =$ `married` $x_k =$ `single`	$a_j + a_k = 1,\ \{a_j, a_k\} \in \{0, 1\}$
Ordinal Feature Encoding	✗	✗	preserve one-hot encoding of `max_degree_BS`, `max_degree_MS`	$x_j =$ `max_degree_BS` $x_k =$ `max_degree_MS`	$a_j = x'_j - x_j \quad a_k = x'_k - x_k$ $x'_j + x'_k = 1 \quad x'_k \geq x'_j,$ $\{x'_j, x'_k\} \in \{0, 1\}$
Logical Implications	✗	✗	if `is_employed` $=$ `TRUE` then `work_hrs_per_week` ≥ 0 else `work_hrs_per_week` $= 0$	$x_j =$ `is_employed`, $x_k =$ `work_hrs_per_week`	$a_j = x'_j - x_j \quad a_k = x'_k - x_k$ $x'_j \in \{0, 1\} \quad x'_k \in [0, 168],$ $x'_k \leq 168\, x'_j$
Deterministic Causal	✗	✗	if `years_at_residence` increases then `age` will increase commensurately	$x_j =$ `years_at_residence` $x_k =$ `age`	$a_j \leq a_k$

Table 1: Catalog of actionability constraints. We show an example of each constraint type to show that they can be expressed in natural language and encoded as a constraint in a combinatorial optimization problem. Here: *Sep.* denotes classes of constraints that are *separable*, meaning that they can be specified for each feature independently; *Cvx.* denotes constraints that are convex. We let $\boldsymbol{x}' := \boldsymbol{x} + \boldsymbol{a}$ when listing the encoding.

(i) the actionability constraints, $\boldsymbol{a} \in A(\boldsymbol{x})$, which could be discrete and non-convex (see Table 1); and $f(\boldsymbol{x} + \boldsymbol{a}) = +1$, which could be difficult to encode for models with complex decision boundaries (e.g., neural networks or random forests).

Recourse Verification The *recourse verification* task seeks to determine if the recourse optimization problem in Eq. (1) is *infeasible*, i.e., to test if $f(\boldsymbol{x} + \boldsymbol{a}) = -1$ for all $\boldsymbol{a} \in A(\boldsymbol{x})$. We formalize the verification procedure as a function such that:

$$\text{Recourse}(\boldsymbol{x}, f, A) = \begin{cases} \texttt{Yes}, & \text{only if there exists } \boldsymbol{a} \in A(\boldsymbol{x}) \text{ such that } f(\boldsymbol{x} + \boldsymbol{a}) = +1 \\ \texttt{No}, & \text{only if } f(\boldsymbol{x} + \boldsymbol{a}) = -1 \text{ for all } \boldsymbol{a} \in A(\boldsymbol{x}) \\ \perp, & \text{otherwise} \end{cases}$$

We say that the procedure *certifies feasibility* from $\boldsymbol{x}$ if it returns `Yes`, and *certifies infeasibility* if it returns `No`. The procedure *abstains* by returning the flag $\perp$ when it cannot certify feasibility or infeasibility. We explain the interpretation of the outcomes next.

Pitfalls in Using Provision for Verification A procedure that solves the recourse provision problem in Eq. (1) can be used for recourse verification when it can *certify infeasibility*. In practice, this requires that the procedure is able to (i) account for all constraints in an action set $A(\boldsymbol{x})$ and (ii) analyze or check every action in the action set. Methods that fail to satisfy these requirements will output results that are incorrect or inconclusive. We describe two concrete kinds of pitfalls that could result in significant harm in applications such as lending or hiring:

Loopholes: A *loophole* is an instance where a method returns an action that violates actionability constraints. In practice, a method for recourse provision returns loopholes if it neglects a subset of actionability constraints. Formally, the procedure solves the recourse problem in Eq. (1) using a relaxed action set $A^{\text{ext}}(\boldsymbol{x}) \supseteq A(\boldsymbol{x})$, and outputs a technically infeasible action $a \in A^{\text{ext}}(\boldsymbol{x}) \setminus A(\boldsymbol{x})$.

Blindspots: A *blindspot* is an instance where a method fails to certify feasibility. Formally, a blindspot arises when there exists an action that provides recourse but a method fails to find it. This may occur when recourse provision method is searching for actions in a way that is not exhaustive and may overlook certain feasible actions.

In Table 13 in the Appendix, we show that these failure modes affect multiple methods for recourse provision, thus not enabling to faithfully check whether recourse is infeasible using standard approaches. This pervasive issue shows the need for treating recourse verification as a special problem on its own.

Specifying Action Sets Semantically meaningful features obey certain rules on *how* they can be changed. In Table 1, we show how to state these conditions in natural language and encode them as constraints in an optimization problem.

Assumptions surrounding actionability differ across individuals and applications [see 75, 5]. Encoding these constraints could be done in a way that promotes transparency, contestability, and participatory design. In particular, individuals could specify their assumptions in natural language, which would enable stakeholders without technical expertise in machine learning to scrutinize and contest these assumptions. In the event that stakeholders cannot reach a consensus, one can certify recourse infeasibility using a *subset* of actionability constraints they agree on. This collection of constraints is never empty, as it always contains a set of *inherent* constraints, e.g., conditions that pertain to how a feature is encoded, or its physical limits (e.g., $\texttt{work\_hrs\_per\_week} \leq 168$).

Use Cases Suppose the model f assigns a *prediction without recourse* to a point $\boldsymbol{x}$. Such predictions may be undesirable in applications where we would like to ensure access (e.g., lending), and desirable in applications where we would like to safeguard against gaming (e.g., content moderation):

Ensuring Access. We verify if a model violates the right to access by testing whether it assigns a negative prediction $y = -1$ (e.g., deny a loan) to a given person with a feature vector $\boldsymbol{x}$ without recourse: $\mathsf{Recourse}(\boldsymbol{x}, f, A) = \texttt{No}$. We set the action set $A(\boldsymbol{x})$ to capture *inherent actionability constraints* that apply to every decision subject. In this case, we flag predictions without recourse even under conservative assumptions as a basic test of infeasibility. The verification procedure can then be used to improve the model or data in order to ensure recourse or robustness for points where recourse is infeasible,

Ensuring Robustness. We certify that the model f is not vulnerable to adversarial manipulations of an example $\boldsymbol{x}$ with a negative prediction $y = -1$ (e.g., content policy violation): $\mathsf{Recourse}(\boldsymbol{x}, f, A) = \texttt{No}$. We specify the action set $A(\boldsymbol{x})$ to capture illegitimate actions [e.g., 22] or to represent an threat model [e.g., 45].

We can minimize the chances that a model will assign a prediction without recourse when we aim to ensure access (respectively, a non-robust prediction when ensuring robustness) by calling verification routines across the model lifecycle. In model development, we would test if a model assigns a fixed prediction over training examples. If this is the case, we can use this information to, e.g., add new actionable features to the dataset, or train models in a way that ensures recourse [68] or robustness [45] for problematic points. In deployment, we would call the procedure for previously unseen points to flag violations at prediction time.

3 Verification with Reachable Sets

We introduce a model-agnostic approach for recourse verification. Our approach stems from the observation that recourse is only feasible in regions of feature space that are delineated by feasible actions. We call these regions *reachable sets* and define them below.

Definition 1 (Reachable Set)**.** *Given a point $\boldsymbol{x}$ and its action set $A(\boldsymbol{x})$, a* reachable set *is a set containing all feature vectors that can be attained by enacting an action $\boldsymbol{a} \in A(\boldsymbol{x})$: $R_A(\boldsymbol{x}) := \{\boldsymbol{x} + \boldsymbol{a} \mid \boldsymbol{a} \in A(\boldsymbol{x})\}$.*

Note that a reachable set of $\boldsymbol{x}$ always includes the point $\boldsymbol{x}$ itself. Reachable sets and action sets capture the same information about actionability using different data structures. In practice, an action set would be stored as a collection of constraints that can be embedded in an optimization problem in compact representations that are applicable to any initial feature vector $\boldsymbol{x}$ (see Table 1 and Fig. 1). In contrast, a reachable set would be stored as feature vectors that satisfy these constraints. Using a reachable set, we can verify recourse by simply checking the predictions of a model for each point in the set.

Concretely, in order to certify recourse feasibility or infeasibility for any model and point $\boldsymbol{x}$, we proceed in two steps. First, given a set of actionability constraints, we generate the reachable set $R = R_A(\boldsymbol{x})$. Second, we query the model at each point in R and certify feasibility if there exists at least one positive prediction, or certify infeasibility if all points in R receive negative predictions. We can also adapt this verification procedure to avoid the need to generate the full reachable set, as we explain shortly.

We formalize the procedure as follows. We are given a model $f : \mathcal{X} \to \mathcal{Y}$ and wish to verify recourse for a point $\boldsymbol{x}$ using the set of points R, where R is either a reachable set $R_A(\boldsymbol{x})$ or its interior approximation $R_A^{\text{int}} \subset R_A(\boldsymbol{x})$. We can express the verification task as a function that checks the predictions of f for each point in R:

$$\textsf{Recourse}(\boldsymbol{x}, f, R) = \begin{cases} \texttt{Yes}, & \text{if there exists } \boldsymbol{x}' \in R \text{ such that } f(\boldsymbol{x}') = +1 \\ \texttt{No}, & \text{if } f(\boldsymbol{x}') = -1 \text{ for all } \boldsymbol{x}' \in R \text{ and } R \supseteq R_A(\boldsymbol{x}) \\ \bot, & \text{if } f(\boldsymbol{x}') = -1 \text{ for all } \boldsymbol{x}' \in R \text{ and } R \subset R_A(\boldsymbol{x}) \end{cases} \tag{2}$$

Reachable sets represent a distinct approach to verification that has several key benefits.

1. **Model-Agnostic Certification**: As reachable sets are constructed from actionability constraints, we can build them once and use them to certify infeasibility for any model. We could also design a hypothetical model-specific approach to recourse verification. Such approach, however, would need to solve a challenging optimization problem in a way that can certify its infeasibility. The problem would also have to satisfy two constraints from Eq. (1): (i) prediction constraints $f(\boldsymbol{x} + \boldsymbol{a}) = +1$, and (ii) actionability constraints $\boldsymbol{a} \in A(\boldsymbol{x})$. Encoding the prediction constraints for complex models such as random forests, gradient boosting tree ensembles, or deep neural networks, can be challenging or impossible for suitable optimization methods that could solve the problem in a certifiable way, e.g. linear programming. Our method avoids this issue as it is applicable to any model.

2. **Safety via Abstention**: The procedure will abstain when it cannot certify recourse. Consider a setting where we call the procedure using an interior approximation of the reachable set $R_A^{\text{int}}(\boldsymbol{x}) \subset R_A(\boldsymbol{x})$ and fail to find a point in $R_A^{\text{int}}(\boldsymbol{x})$ that achieves the target prediction. In this case, the procedure would abstain, and the resulting abstention would flag $\boldsymbol{x}$ as a *potential* prediction without recourse. This property has practical benefits because it allows us to make use of interior approximations of reachable sets without

yielding results that are incorrect or inconclusive. For example, we can call $\mathsf{Recourse}(\boldsymbol{x}, f, R)$ with an interior reachable set $R = R_A^{\text{int}}(\boldsymbol{x})$ to screen points that have recourse. We can then revisit those points where the procedure abstained with a better approximation or the full reachable set $R = R_A(\boldsymbol{x})$.

Next, we discuss special cases that can make the verification with reachable sets simpler of more efficient.

3.1 Divide and Conquer Verification using Action Set Partitioning

Depending on the nature of the actionability constraints and the feature space, reachable sets can potentially grow large. In practical applications where recourse verification is useful this is not necessarily the case (see Section 5). However, it can be helpful to simplify the verification problem for computational reasons. In addition to verifying recourse with an inner approximation of the reachable set, as explained before, we can also partition the actions in a way that speeds up the verification.

For this, observe that actions on some features can be independent of others. For instance, take `prior_applicant` and `age` ≥ 60 from Fig. 1. A change to one of these features can be enacted without any dependence on the other one. Formally, we say that two subsets of features $S, T \subseteq [1, \ldots, d]$ are *separable* if $A(\boldsymbol{x}) = A_S(\boldsymbol{x}) \oplus A_T(\boldsymbol{x})$, where $A_U(\boldsymbol{x}) \subseteq A(\boldsymbol{x})$ is a subset of actions that only modify features in U – i.e., for any $\boldsymbol{a} \in A_U(\boldsymbol{x})$ it holds that if $a_i \neq 0$ then $i \in U$, and $U \oplus V := \{\mathbf{u} + \mathbf{v} \mid \mathbf{u} \in U, \mathbf{v} \in V\}$. In other words, one can reconstruct the full action set by summing any action that only modifies features in S and any action that modifies features in T. We can use separability to partition the action set:

Definition 2 (Action Set Partition). *A partition $\mathcal{S}$ is a collection of feature subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, with each $S_j \subseteq [1, \ldots, d]$, such that the action set decomposes as $A(\boldsymbol{x}) = \bigoplus_{S \in \mathcal{S}} A_S(\boldsymbol{x})$.*

Identifying such separable features and partitioning the action set as finely as possible enables us to significantly speed up concrete algorithms for recourse verification in practice. For each feature subset in a partition $S_j \in \mathcal{S}$, we can generate the respective reachable set or its inner approximation $R_j \subseteq R_{A_{S_j}}(\boldsymbol{x})$. We can leverage the partitioning by modifying the verification procedure as follows:

$$\mathsf{Recourse}(\boldsymbol{x}, f, R_A(\boldsymbol{x})) = \begin{cases} \texttt{Yes}, & \text{if there exists } j \in [1, \ldots, k] \text{ such that } \mathsf{Recourse}(\boldsymbol{x}, f, R_j) = \texttt{Yes} \\ \texttt{No}, & \text{if } \mathsf{Recourse}(\boldsymbol{x}, f, R_j) = \texttt{No} \text{ for all } j \in [1, \ldots, k] \\ \bot, & \text{otherwise} \end{cases},$$

where $\mathsf{Recourse}(\boldsymbol{x}, f, R_j)$ would abstain analogously to Eq. (2): $\mathsf{Recourse}(\boldsymbol{x}, f, R_j) = \bot$ if $f(\boldsymbol{x}) = -1$ for all $\boldsymbol{x} \in R_j$ and if $R_j \subset R_{A_{S_j}}(\boldsymbol{x})$ is an inner approximation of the respective partition of the reachable set.

In the worst case, the size of the reachable set could grow exponentially with the number of features as $O(c^d)$, where $c > 1$. Thus, denoting the size of the largest feature subset in a partition as $t = \max_{S \in \mathcal{S}} |S|$, this procedure cuts down the worst-case runtime from $O(c^d) = O(c^{k \cdot t})$ to $O(k \cdot c^t)$.

3.2 Fixed Points and Regions

Certain classes of reachable sets can additionally support verification:

Definition 3 (Fixed Point). *A point $\boldsymbol{x}$ is* fixed *if its reachable set only contains itself: $R_A(\boldsymbol{x}) = \{\boldsymbol{x}\}$.*

Fixed points can occur in clusters. If there exist separable features in $A(\boldsymbol{x})$, and some of them are immutable, then $\boldsymbol{x}$ has a set of *sibling points* $N_A(\boldsymbol{x})$: all points that contain modifications of the immutable features of $\boldsymbol{x}$. If $\boldsymbol{x}$ is a fixed point, then every sibling $\boldsymbol{x}' \in N_A(\boldsymbol{x})$ is also fixed.

Definition 4 (Fixed Region). *A* fixed region *is a reachable set $R_A(\boldsymbol{x})$ such that for any $\boldsymbol{x}' \in R_A(\boldsymbol{x})$ we have $R_A(\boldsymbol{x}') = R_A(\boldsymbol{x})$.*

Given a fixed point, we can determine if a model violates recourse by checking its prediction on a single point. Given a fixed region, we can verify recourse for all points within it without generating additional reachable sets.

Fixed points arise under common actionability constraints. For instance:

Proposition 5. *Any classification task with bounded features whose actions obey monotonicity constraints must contain at least one fixed point.*

Fixed points and regions exhibit compositional properties if one extends the dataset with new features.

Proposition 6. *Consider adding a new feature $\mathcal{X}_{d+1} \subseteq \mathbb{R}$ to a set of d features $\mathcal{X} \subseteq \mathbb{R}^d$. Let $A_{d+1}(x_{d+1})$ be the actions from a point with feature value $x_{d+1} \in \mathcal{X}_{d+1}$. Any fixed point $\boldsymbol{x} \in \mathcal{X}$ induces the following confined regions in the $(d+1)$-dimensional space:*

- *$|\mathcal{X}_{d+1}|$ fixed points feature space, if x_{d+1} is immutable.*
- *A fixed point $\boldsymbol{z}_0 := (\boldsymbol{x}, v_{d+1})$ where v_{d+1} is an extreme point of $\mathcal{X}_{d+1}$.*
- *A fixed region if $A_{d+1}(x_{d+1}) = A_{d+1}(x'_{d+1})$ for any $x_{d+1}, x'_{d+1} \in \mathcal{X}_{d+1}$.*

Proposition 6 provides a cautionary result. For instance, suppose that we have identified fixed points. To enable recourse for them, we could choose to source additional features. In this case, we must take care that these new features remove the fixed points and regions and not only propagate them.

3.3 Prevalence-Based Certification

Finally, we show that given a set of labeled examples, reachable sets enable us to verify recourse for any classifier under certain assumptions on its performance.

Theorem 7 (Certification with Labeled Examples). *Suppose we have a dataset of labeled examples $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Every model $f : \mathcal{X} \to \mathcal{Y}$ can provide recourse to $\boldsymbol{x}$ if:*

$$\mathsf{FNR}(f) < \frac{1}{n^+} \sum_{i=1}^{n} \mathbb{1}[\boldsymbol{x}_i \in R \ \wedge \ y_i = +1] \tag{3}$$

where $\mathsf{FNR}(f) := \frac{1}{n^+} \sum_{i=1}^n \mathbb{1}[f(\boldsymbol{x}_i) = -1 \ \wedge \ y_i = +1]$ is the false negative rate of f on $\mathcal{D}$ and where n^+ is number of positive examples in $\mathcal{D}$, and $R \subseteq R_A(\boldsymbol{x})$ is any subset of the reachable set.

The proof of Theorem 7 relies on the pigeonhole principle, and is provided in Appendix B. The theorem states that any classifier that is sufficiently accurate on a given dataset must provide recourse for a given point. The result applies to any sample of points and any model, and ties together accuracy, feasibility, and reachable sets in a general way. Its practical application, however, requires datasets that are "dense" enough so that a reachable set contains other points in the dataset.

4 Algorithms

In this section, we present a toolbox of algorithms for recourse verification with reachable sets. Our algorithms are designed to attack the problem of recourse verification from different angles in order to

Name	Purpose	Action Space	Usage for Verification
FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$	Check if $\boldsymbol{x}$ is a fixed point	Disc., cont.	If $\boldsymbol{x}$ is fixed, Recourse$(\boldsymbol{x}, f, A) =$ `Yes` if $f(x) = +1$, and `No` otherwise
IsReachable$(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x}))$	Check if $\boldsymbol{x}'$ is reachable from $\boldsymbol{x}'$	Disc., cont.	If $\boldsymbol{x}'$ is in a dataset with label $y = +1$, Recourse$(\boldsymbol{x}, f, A) =$ `Yes`.
GetReachableSet$(\boldsymbol{x}, A(\boldsymbol{x}))$	Generate reachable set of $\boldsymbol{x}$	Disc.	See Eq. (2)
SampleAudit$(S, A(\cdot))$	Generate reachable sets for a sample of points	Disc.	Apply Eq. (2) to each reachable set

Table 2: A toolbox of algorithms for recourse verification with reachable sets.

minimizes abstentions – i.e., we wish to confirm as many predictions that have recourse and to detect as many predictions without recourse as possible, applying one or several methods within the toolbox. To this end, they can detect fixed points, construct reachable sets over discrete feature spaces, and identify predictions with recourse by testing reachability over samples. Table 2 provides a summary.

Fixed Point Detection We start with a method to detect fixed points, which we use as a standalone method as well as a building block in later methods. This method verifies if $\boldsymbol{x}$ is a fixed point by solving the optimization problem:

$$\begin{aligned} \text{FindAction}(\boldsymbol{x}, A(\boldsymbol{x})) \in \quad & \operatorname{argmin} \quad \|\boldsymbol{a}\| \\ & \text{s.t.} \quad \boldsymbol{a} \in A(\boldsymbol{x}) \setminus \{\mathbf{0}\}. \end{aligned}$$

If FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ is infeasible, we know that $\boldsymbol{x}$ is a fixed point. We formulate FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ as a mixed integer program, and solve it with an off-the-shelf solver [see e.g., 25]. We present a formulation that can encode all actionability constraints from Table 1 in Appendix C. Given a point $\boldsymbol{x}$ is fixed, we can certify Recourse$(\boldsymbol{x}, f, A) =$ `Yes` if $f(\boldsymbol{x}) = +1$ and `No` if $f(\boldsymbol{x}) = -1$.

Due to improvements in hardware, software, and research over the last three decades [7, 6], off-the-shelf solvers can find solutions to optimization problems such as FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ one efficiently in less than a second. Still, we can speed up this procedure by leveraging action set separability, analogously to the procedure in Section 3.1.

Verification on Observed Data When verifying recourse for multiple points, we can avoid verification for points that have previously appeared in reachable sets. The next method is designed to certify that a point has recourse by testing if a point $\boldsymbol{x}$ can reach another point $\boldsymbol{x}' \in R_A(\boldsymbol{x})$, e.g., in a dataset, that is assigned a positive prediction:

$$\begin{aligned} \text{IsReachable}(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x})) := \quad & \min \quad 1 \\ & \text{s.t.} \quad \boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{a} \\ & \qquad \boldsymbol{a} \in A(\boldsymbol{x}) \end{aligned}$$

The $\min 1$ notation denotes a feasibility problem. We formulate this problem as a mixed integer program and solve it using an off-the-shelf solver. Given a set of positive samples S^+, we can call this routine for all points $\boldsymbol{x}' \in S^+$. This will screen out points $\boldsymbol{x}$ that have recourse. Then, we can certify recourse Recourse$(\boldsymbol{x}, f, A) =$ `Yes` for these points.

Reachable Set Generation Our next method can certify feasibility and infeasibility in a discrete feature space by constructing a reachable set. We present the procedure for generating a reachable set of a given point $\boldsymbol{x}$ in Algorithm 1. For this, we repeatedly solve FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ (line 3), while removing the

Algorithm 1 GetReachableSet

Require: $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is discrete; $A(\boldsymbol{x})$

$R \leftarrow \{\boldsymbol{x}\}, F \leftarrow A(\boldsymbol{x})$

1: **repeat**
2: **if** FindAction$(\boldsymbol{x}, F)$ is feasible **then**
3: $\boldsymbol{a}^* \leftarrow$ FindAction$(\boldsymbol{x}, F)$
4: $R \leftarrow R \cup \{\boldsymbol{x} + \boldsymbol{a}^*\}$
5: $F \leftarrow F \setminus \{\boldsymbol{a}^*\}$
6: **until** stopping condition

Output $R \subseteq R_A(\boldsymbol{x})$

Algorithm 2 SampleAudit

Require: Sample $S = \{\boldsymbol{x}_i\}_{i=1}^n$ and $A(\cdot)$

1: $\mathbb{C} \leftarrow \{\ \}$
2: **repeat**
3: $\boldsymbol{x}_i \leftarrow$ Pop(S)
4: $R_i \leftarrow$ GenReachableSet$(\boldsymbol{x}_i, A(\boldsymbol{x}_i))$
5: **if** $R_i = \{\boldsymbol{x}_i\}$ **then**
6: $S \leftarrow S \setminus N_A(\boldsymbol{x}_i, A(\boldsymbol{x}_i))$
7: **else if** R_i is a fixed region **then**
8: $S \leftarrow S \setminus R_i$
9: $\mathbb{C} \leftarrow \mathbb{C} \cup \{R_i\}$
10: **until** no points remain in S

Output $\mathbb{C}$, collection of reachable sets

previous solution from the considered action set at every next step (line 5). The procedure continues until the problem becomes infeasible or another stopping condition is met. Once a reachable sets is constructed, it can be reused to verify recourse feasibility for any model. As described in Section 3, we might be interested in generating only a subset of the reachable set $R_A^{\text{int}}(\boldsymbol{x}) \subset R_A(\boldsymbol{x})$. In this case, the stopping condition could be that the algorithm has identified a certain number of points in the reachable set.

Full Sample Audit In Algorithm 2, we present an algorithm to produce a collection of reachable sets for each point in a dataset – i.e., a collection that can be used to perform the verification procedure in Eq. (2). Our procedure seeks to reduce the time needed to build these reachable sets by exploiting the properties of fixed points and fixed regions in Section 3. For instance, if a reachable set is a fixed region, then by definition we do not need to generate reachable sets for any other point in the dataset that also falls into the fixed region. We detail the full auditing procedure with optimizations Appendix C.

5 Experiments

We present an empirical study of infeasibility in recourse. Our goal is to study the prevalence of predictions without recourse under actionability constraints, and to characterize the reliability of verification using existing methods.

Setup We work with three publicly available lending datasets in Table 3. Each dataset pertains to a task where predictions without recourse preclude access to credit and recourse provision. We use each dataset to fit a classifier using *logistic regression* (LR) and *XGBoost* (XGB). We construct reachable sets for each training example using Algorithm 1 in CPLEX v22.1 on a 3.2GHz CPU with 8GB RAM. We benchmark our method (Reach) against baseline methods for recourse provision: AR [73], a *model-specific* method that can certify infeasibility for linear models with separable actionability constraints; DiCE [58], a *model-agnostic* method that supports separable actionability constraints.

We evaluate the feasibility of recourse under nested action sets: Non-Separable, which includes constraints on immutability, monotonicity, and non-separable constraints such as causal relations; Separable, which includes constraints on immutability and monotonicity; Simple, which includes constraints on immutability. To certify feasibility, we use standard discretization into bins of continuous features. We summarize the

Dataset	Model Type	Metrics	Simple			Separable			Non-Separable		
			Reach	AR	DiCE	Reach	AR	DiCE	Reach	AR	DiCE
`german` Dua and Graff [19] $n = 1,000$ $d = 24$	LR	Recourse	90.8%	99.1%	94.7%	91.9%	95.5%	82.7%	94.9%	22.4%	11.3%
		No Recourse	0.0%	0.9%	—	0.4%	4.5%	—	2.1%	4.5%	—
		Abstain	9.2%	—	—	7.7%	—	—	3.0%	—	—
		Loopholes	—	0.0%	0.0%	—	0.0%	0.0%	—	73.1%	71.4%
		Blindspots	—	0.0%	5.3%	—	0.0%	17.3%	—	0.0%	17.3%
	XGB	Recourse	90.4%		94.7%	91.9%		84.5%	94.9%		21.3%
		No Recourse	0.0%		—	0.4%		—	2.1%		—
		Abstain	9.6%	—	—	7.7%	—	—	3.0%	—	—
		Loopholes	—		0.0%	—		0.0%	—		62.8%
		Blindspots	—		5.3%	—		15.5%	—		16.0%
`givemecredit` Kaggle [37] $n = 8,000$ $d = 13$	LR	Recourse	100.0%	100.0%	100.0%	99.9%	99.9%	99.9%	99.9%	64.9%	53.6%
		No Recourse	0.0%	0.0%	—	0.0%	0.1%	—	0.1%	0.0%	—
		Abstain	0.0%	—	—	0.1%	—	—	0.1%	—	—
		Loopholes	—	0.0%	0.0%	—	0.0%	0.0%	—	35.1%	46.4%
		Blindspots	—	0.0%	0.0%	—	0.0%	0.1%	—	0.0%	0.0%
	XGB	Recourse	100.0%		100.0%	99.9%		99.9%	99.9%		28.6%
		No Recourse	0.0%		—	0.0%		—	0.1%		—
		Abstain	0.0%	—	—	0.1%	—	—	0.1%	—	—
		Loopholes	—		0.0%	—		0.0%	—		71.4%
		Blindspots	—		0.0%	—		0.1%	—		0.0%
`heloc` FICO [21] $n = 3,184$ $d = 29$	LR	Recourse	24.7%	85.2%	51.7%	29.0%	62.2%	48.2%	57.3%	23.5%	20.4%
		No Recourse	0.0%	14.8%	—	0.0%	37.8%	—	41.9%	37.8%	—
		Abstain	75.3%	—	—	71.0%	—	—	0.8%	—	—
		Loopholes	—	0.0%	0.0%	—	0.0%	0.0%	—	38.8%	27.8%
		Blindspots	—	0.0%	48.3%	—	0.0%	51.8%	—	0.0%	51.8%
	XGB	Recourse	84.4%		99.8%	35.1%		57.5%	73.2%		23.4%
		No Recourse	0.0%		—	0.0%		—	26.4%		—
		Abstain	15.6%	—	—	64.9%	—	—	0.5%	—	—
		Loopholes	—		0.0%	—		0.0%	—		34.1%
		Blindspots	—		0.2%	—		42.5%	—		42.5%

Table 3: Reliability of recourse of verification over datasets, model classes, and actionability constraints. We determine the feasibility of recourse for training examples using our method (Reach), then use them to evaluate the reliability of verification using salient methods for recourse provision. We report the following metrics for each method and model type: *Recourse* – % of points where a method certifies that recourse exists, *No Recourse* – % of points where the method certifies no recourse exists, *Abstain* – % of points in which Reach cannot determine if recourse exists, *Loopholes* – % of points whose actions are unactionable, *Blindspots* – % of points where a method fails to return an action.

reliability of recourse verification for each dataset, method, and model class in Table 3. In what follows, we discuss these results in detail.

On Predictions without Recourse Our results show how recourse may not exist under actionability constraints. Recourse is vacuously feasible under simple constraints such as immutability and integrality, and infeasible once we consider more complex constraints. On `german`, for example, LR only assigns predictions without recourse under Separable and Non-Separable constraints to 0.4% and 2.1% of the data, respectively.

We find minor variations in the prevalence of predictions without recourse across model classes. Given that the reachable sets do not change under a fixed dataset and actionability constraints, these differences reflect differences in the number of negative predictions across models. We observe that predictions without recourse can drastically change across model types that are equally accurate at a population level. In Non-Separable `heloc`, for example, we observe a 15.5% difference in predictions without recourse between LR and XGB even though both classifiers have similar performance (Test AUC of 0.729 vs 0.737). This highlights the potential to choose between models to ensure recourse.

On Pitfalls of Verification Our results highlight common failure modes in using methods for recourse provision for verification described in Section 2. In `heloc`, for example, we observe 26.4% of points without recourse with XGB. In cases where recourse is feasible, methods may fail to return any actions. However, we may be unable to tell if a point has recourse or if a method was unable to generate it in the first place. For example, DiCE fails to find actions for 42.5% of points. However, there may exist feasible actions. This effect highlights the potential failure to account for actionability constraints by, e.g., post-hoc filtering [50]. In this case, DiCE cannot produce any actions after filtering a set of diverse counterfactual explanations to enforce implication constraints related to deterministic causal relationships and a thermometer encoding.

On the Value of Model-Agnostic Verification Although methods such as AR can certify infeasibility for linear models under separable constraints, they start yielding loophole actions as soon as the constraints become more complex. For instance, in `heloc`, AR flags 37.8% of predictions as without recourse with Separable, but overlooks an additional 4.1% of predictions without recourse under Non-Separable constraints. Reach does have limitations in verifying recourse when it hits a predefined hard limit for building out the full reachable that we set to 25 points for these experiments. Although the number of abstentions can be reduced by increasing the hard limit, we can also certify more points by using the methods in conjunction with one another. For instance, for points where Reach abstains, AR can be used in non-separable cases to determine the feasibility of a point, and if DiCE can find actions for an individual, Reach can determine if the action is feasible. In cases where DiCE or AR cannot find actions for constraints that are non-separable, Reach can be used to determine if the point is fixed or if we indeed observe recourse infeasibility.

6 Demonstrations

We demonstrate how our methods can ensure recourse in lending. We include an additional study to show how we can evaluate adversarial robustness in content moderation in Appendix E.

Setup We work with the FICO `heloc` dataset [21], which covers $n = 3\,184$ consumers and contains $d = 29$ features about their credit history. Here, $y_i = +1$ if a consumer i has duly repaid a home equity loan. Our goal is to ensure recourse over the training data – so that we can flag models that permanently deny access to credit [44, 12], and use recourse provision methods to produce adverse action notices [1]. We work with a domain expert in the U.S. credit industry to identify common constraints on features. Our final action set includes 24 constraints, both separable (e.g., `RevolvingTradesWBalance` is a positive integer, `MostRecentTradeLastYear` can only increase), and non-separable (e.g., `RevolvingDebtBurdenLeq30`, `RevolvingDebtBurdenGeq60`).

Results We generate reachable sets for all points in the training data using Algorithm 1 and use them to perform recourse verification for LR and XGB classifiers. We summarize the feasibility of recourse in

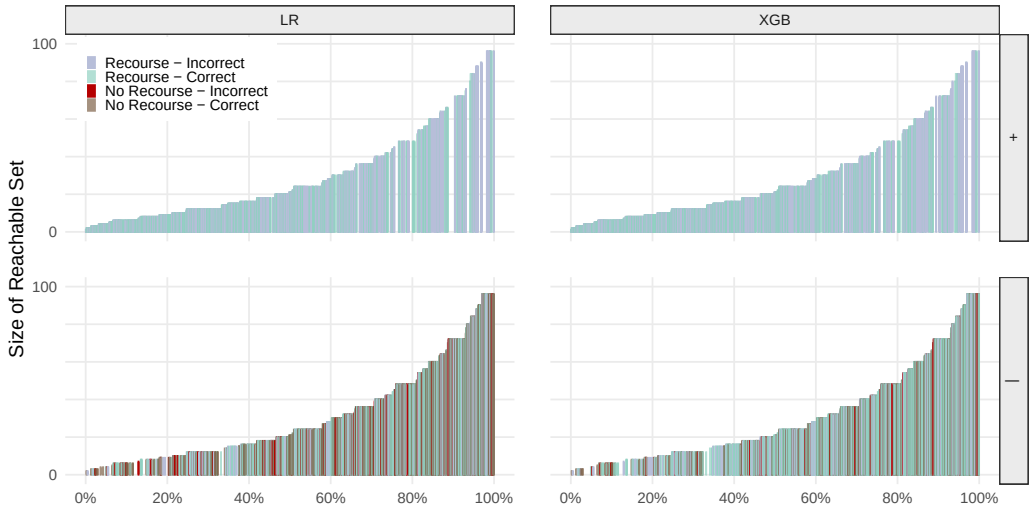

Figure 2: Composition of reachable sets for the `heloc` for LR and XGB. Each plot shows the size of reachable sets for each training example delineated by Algorithm 1. The top row displays sizes of reachable sets for samples with negative predictions and the bottom row for samples with positive predictions. Correct/incorrect denotes where the true label does/does not equal the predicted label and Recourse/No Recourse denotes if recourse is feasible/infeasible. We highlight predictions without recourse for both correctly classified and incorrectly classified negative points. We can see predictions without recourse are prevalent with all reachable set sizes and can be drastically different between classifiers.

Fig. 5. Our results reveal 733 predictions without recourse for LR, and 453 for XGB. In this case, we find 5 fixed points that are assigned positive predictions. Thus, all predictions without recourse stem from a generalized reachable set. The mix of individual feature constraints and constraints on the interactions between features causes reachable sets with no recourse.

Predictions without recourse may have serious implications for consumers attempting to acquire a loan. A specific example is a consumer with 10 years of account history, and 15 open credit and fixed loans with a majority of them paid off. Among their open fixed loans, there is a substantial remaining balance that needs to be paid, and they experienced a delinquent trade within the past year. They are denied by both classifiers. This consumer has the ability to reach 7 other points by reducing their one credit card loan with balance and increasing the number of years they have open and active loans. However, even if with all these changes, they will still be denied approval.

Our results can guide interventions in model development to ensure recourse. At a minimum, practitioners can use the information from this analysis for model selection. In this case, we find that both classifiers have similar performance in terms of AUC, but XGB assigns 280 fewer predictions without recourse. More generally, we can identify immutable features that lead to infeasibility in predictions. In this case, our analysis reveals that a key feature among individuals assigned predictions without recourse is `MaxDelqEver`, which determines the maximum duration of delinquency. In this case, one can restore recourse by replacing this feature with an alternative that is mutable `MaxDelqInLast5Years`.

7 Concluding Remarks

Our work highlights the inherent value of verifying infeasibility in recourse as a tool to ensure access in consumer-facing applications. The normative basis for access stems from principles of equality of

opportunity (e.g., in hiring) and universal access (e.g., in affordable healthcare). In such settings, we would want a model to ensure access – even if this comes at a cost – because it reflects the kind of society we want to build.

Ensuring access may benefit all stakeholders in consumer-facing applications. In lending, for example, we only observe labels for consumers who are assigned a specific prediction [due to selective labeling 49, 15, 81]. Thus, consumers assigned predictions without recourse cannot generate labeled examples to signal their creditworthiness [12]. In the United States, these effects have cut off credit access for large consumer segments whose creditworthiness is not known [see, e.g., 26M "credit invisibles" 82].

Limitations Given that actionability varies across individuals as well as subpopulations, verification should be used as a test to flag models that will preclude access, rather than as a tool to "rubber-stamp" for safety. Our treatment of infeasibility does not consider probabilistic causal effects – e.g., when changing a feature may induce changes in other features as per a probabilistic causal model [see 43, 48, 16, 78]. Our machinery may be useful for modeling complex interactions in these settings. However, infeasibility requires a probabilistic definition that can characterize the "reducible" uncertainty due to model development from the uncertainty in the underlying causal process.

References

[1] 2013. URL https://www.consumercomplianceoutlook.org/2013/second-quarter/adverse-action-notice-requirements-under-ecoa-fcra/.

[2] Alon, Tal, Magdalen Dobson, Ariel Procaccia, Inbal Talgam-Cohen, and Jamie Tucker-Foltz. Multiagent evaluation mechanisms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1774–1781, 2020.

[3] Altmeyer, Patrick, Giovan Angela, Aleksander Buszydlik, Karol Dobiczek, Arie Deursenvan , and Cynthia Liem. Endogenous macrodynamics in algorithmic recourse. In *First IEEE Conference on Secure and Trustworthy Machine Learning*.

[4] Arneson, Richard. Equality of Opportunity. In Zalta, Edward N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2015 edition, 2015.

[5] Barocas, Solon, Andrew D Selbst, and Manish Raghavan. The hidden assumptions behind counterfactual explanations and principal reasons. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 80–89, 2020.

[6] Bixby, Robert and Edward Rothberg. Progress in computational mixed integer programming a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, 2007.

[7] Bixby, Robert E, Mary Fenelon, Zonghao Gu, Edward Rothberg, and Robert Wunderling. Mixed integer programming: a progress report. *The Sharpest Cut*, pages 309–326, 2004.

[8] Bogen, Miranda and Aaron Rieke. Help wanted: An examination of hiring algorithms, equity, and bias. *Upturn, December*, 7, 2018.

[9] Calzavara, Stefano, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. Treant: training evasion-aware decision trees. *Data Min. Knowl. Discov.*, 2020.

[10] Chen, Yatong, Jialu Wang, and Yang Liu. Strategic recourse in linear classification. *arXiv preprint arXiv:2011.00355*, 2020.

[11] Cheng, Furui, Yao Ming, and Huamin Qu. Dece: Decision explorer with counterfactual explanations for machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1438–1447, 2020.

[12] Chien, Jennifer, Margaret Ew Roberts, and Berk Ustun. Learning through Recourse under Censoring. *NeurIPS Workshop on Learning and Decision-Making with Strategic Feedback*, 2021.

[13] Cui, Zhicheng, Wenlin Chen, Yujie He, and Yixin Chen. Optimal action extraction for random forests and boosted trees. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 179–188, 2015.

[14] Daniels, Norman. Equity of access to health care: some conceptual and ethical issues. *The Milbank Memorial Fund Quarterly. Health and Society*, pages 51–81, 1982.

[15] De-Arteaga, Maria, Artur Dubrawski, and Alexandra Chouldechova. Learning under selective labels in the presence of expert consistency. *arXiv preprint arXiv:1807.00905*, 2018.

[16] Dominguez-Olmedo, Ricardo, Amir H Karimi, and Bernhard Schölkopf. On the adversarial robustness of causal algorithmic recourse. In *International Conference on Machine Learning*, pages 5324–5342. PMLR, 2022.

[17] Dong, Jinshuo, Aaron Roth, Zachary Schutzman, Bo Waggoner, and Zhiwei Steven Wu. Strategic classification from revealed preferences. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 55–70. ACM, 2018.

[18] Downs, Michael, Jonathan L Chu, Yaniv Yacoby, Finale Doshi-Velez, and Weiwei Pan. Cruds: Counterfactual recourse using disentangled subspaces. *ICML WHI*, 2020:1–23, 2020.

[19] Dua, Dheeru and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[20] Eubanks, Virginia. *Automating inequality: How high-tech tools profile, police, and punish the poor*. St. Martin's Press, 2018.

[21] FICO. Fico heloc, 2018. URL https://community.fico.com/s/explainable-machine-learning-challenge.

[22] Ghalme, Ganesh, Vineet Nair, Itay Eilat, Inbal Talgam-Cohen, and Nir Rosenfeld. Strategic classification in the dark. In *International Conference on Machine Learning*, pages 3672–3681. PMLR, 2021.

[23] Gilani, Zafar, Ekaterina Kochmar, and Jon Crowcroft. Classification of twitter accounts into automated agents and human users. In *Proceedings of the 2017 IEEE/ACM international conference on advances in social networks analysis and mining 2017*, pages 489–496, 2017.

[24] Gilman, Michele E. Poverty lawgorithms: A poverty lawyer's guide to fighting automated decision-making harms on low-income communities. *Data & Society*, 2020.

[25] Gleixner, Ambros, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

[26] Guo, Hangzhi, Feiran Jia, Jinghui Chen, Anna Squicciarini, and Amulya Yadav. Rocoursenet: Distributionally robust training of a prediction aware recourse model. *arXiv preprint arXiv:2206.00700*, 2022.

[27] Gupta, Vivek, Pegah Nokhiz, Chitradeep Dutta Roy, and Suresh Venkatasubramanian. Equalizing recourse across groups. *arXiv preprint arXiv:1909.03166*, 2019.

[28] Hardt, Moritz, Nimrod Megiddo, Christos Papadimitriou, and Mary Wootters. Strategic classification. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 111–122. ACM, 2016.

[29] Hardt, Moritz, Eric Mazumdar, Celestine Mendler-Dünner, and Tijana Zrnic. Algorithmic collective action in machine learning. *arXiv preprint arXiv:2302.04262*, 2023.

[30] Harris, Keegan, Hoda Heidari, and Steven Z Wu. Stateful strategic regression. *Advances in Neural Information Processing Systems*, 34:28728–28741, 2021.

[31] Harris, Keegan, Valerie Chen, Joon Kim, Ameet Talwalkar, Hoda Heidari, and Steven Z Wu. Bayesian persuasion for algorithmic recourse. *Advances in Neural Information Processing Systems*, 35:11131–11144, 2022.

[32] Heidari, Hoda, Vedant Nanda, and Krishna Gummadi. On the long-term impact of algorithmic decision policies: Effort unfairness and feature segregation through social learning. In *International Conference on Machine Learning*, pages 2692–2701. PMLR, 2019.

[33] Hein, Matthias and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2017.

[34] Hurley, Mikella and Julius Adebayo. Credit scoring in the era of big data. *Yale JL & Tech.*, 18:148, 2016.

[35] Joshi, Shalmali, Oluwasanmi Koyejo, Warut Vijitbenjaronk, Been Kim, and Joydeep Ghosh. Towards realistic individual recourse and actionable explanations in black-box decision making systems. *arXiv preprint arXiv:1907.09615*, 2019.

[36] Jurgovsky, Johannes, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen. Sequence classification for credit-card fraud detection. *Expert Systems with Applications*, 100:234–245, 2018.

[37] Kaggle. Give Me Some Credit. http://www.kaggle.com/c/GiveMeSomeCredit/, 2011.

[38] Kanamori, Kentaro, Takuya Takagi, Ken Kobayashi, and Yuichi Ike. Counterfactual explanation trees: Transparent and consistent actionable recourse with decision trees. In *International Conference on Artificial Intelligence and Statistics*, pages 1846–1870. PMLR, 2022.

[39] Kantchelian, Alex, J. D. Tygar, and Anthony D. Joseph. Evasion and hardening of tree ensemble classifiers. In *ICML*, 2016.

[40] Karimi, Amir-Hossein, Gilles Barthe, Borja Balle, and Isabel Valera. Model-agnostic counterfactual explanations for consequential decisions. In *International Conference on Artificial Intelligence and Statistics*, pages 895–905. PMLR, 2020.

[41] Karimi, Amir-Hossein, Julius Von Kügelgen, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. *Advances in neural information processing systems*, 33: 265–277, 2020.

[42] Karimi, Amir-Hossein, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. 2021.

[43] Karimi, Amir-Hossein, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse: from counterfactual explanations to interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 353–362, 2021.

[44] Kilbertus, Niki, Manuel Gomez-Rodriguez, Bernhard Schölkopf, Krikamol Muandet, and Isabel Valera. Improving consequential decision making under imperfect predictions. *arXiv preprint arXiv:1902.02979*, 2019.

[45] Kireev, Klim, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. In *Network and Distributed System Security (NDSS) Symposium*, 2023.

[46] Kleinberg, Jon and Manish Raghavan. How Do Classifiers Induce Agents To Invest Effort Strategically? *ArXiv e-prints*, art. arXiv:1807.05307, July 2018.

[47] Kleinberg, Jon and Manish Raghavan. How do classifiers induce agents to invest effort strategically? *ACM Transactions on Economics and Computation (TEAC)*, 8(4):1–23, 2020.

[48] König, Gunnar, Timo Freiesleben, and Moritz Grosse-Wentrup. Improvement-focused causal recourse (icr). *arXiv preprint arXiv:2210.15709*, 2022.

[49] Lakkaraju, Himabindu, Jon Kleinberg, Jure Leskovec, Jens Ludwig, and Sendhil Mullainathan. The selective labels problem: Evaluating algorithmic predictions in the presence of unobservables. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 275–284, 2017.

[50] Laugel, Thibault, Adulam Jeyasothy, Marie-Jeanne Lesot, Christophe Marsala, and Marcin Detyniecki. Achieving diversity in counterfactual explanations: a review and discussion. *arXiv preprint arXiv:2305.05840*, 2023.

[51] Levanon, Sagi and Nir Rosenfeld. Strategic classification made practical. In *International Conference on Machine Learning*, pages 6243–6253. PMLR, 2021.

[52] Lowd, Daniel and Christopher Meek. Adversarial learning. In *ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647, 2005.

[53] Mahajan, Divyat, Chenhao Tan, and Amit Sharma. Preserving causal constraints in counterfactual explanations for machine learning classifiers. *arXiv preprint arXiv:1912.03277*, 2019.

[54] Makar, Maggie, Ben Packer, Dan Moldovan, Davis Blalock, Yoni Halpern, and Alexander D'Amour. Causally motivated shortcut removal using auxiliary labels. In *International Conference on Artificial Intelligence and Statistics*, pages 739–766. PMLR, 2022.

[55] Mathov, Yael, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous tabular data and adversarial examples. *Knowledge-Based Systems*, 242:108377, 2022.

[56] Miller, John, Smitha Milli, and Moritz Hardt. Strategic classification is causal modeling in disguise. In *International Conference on Machine Learning*, pages 6917–6926. PMLR, 2020.

[57] Milli, Smitha, John Miller, Anca D Dragan, and Moritz Hardt. The social cost of strategic classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 230–239, 2019.

[58] Mothilal, Ramaravind K, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 607–617, 2020.

[59] Nguyen, Duy, Ngoc Bui, and Viet Anh Nguyen. Feasible recourse plan via diverse interpolation. In *International Conference on Artificial Intelligence and Statistics*, pages 4679–4698. PMLR, 2023.

[60] O'Brien, Andrew and Edward Kim. Toward multi-agent algorithmic recourse: Challenges from a game-theoretic perspective. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

[61] Parmentier, Axel and Thibaut Vidal. Optimal counterfactual explanations in tree ensembles. In *International Conference on Machine Learning*, pages 8422–8431. PMLR, 2021.

[62] Pawelczyk, Martin, Teresa Datta, Johannes Heuvelvan-den , Gjergji Kasneci, and Himabindu Lakkaraju. Algorithmic recourse in the face of noisy human responses. *arXiv preprint arXiv:2203.06768*, 2022.

[63] Poyiadzi, Rafael, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. Face: feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 344–350, 2020.

[64] Quinzan, Francesco, Cecilia Casolo, Krikamol Muandet, Niki Kilbertus, and Yucen Luo. Learning counterfactually invariant predictors. *arXiv preprint arXiv:2207.09768*, 2022.

[65] Raghavan, Manish, Solon Barocas, Jon Kleinberg, and Karen Levy. Mitigating bias in algorithmic hiring: Evaluating claims and practices. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 469–481, 2020.

[66] Ramakrishnan, Goutham, Yun Chan Lee, and Aws Albarghouthi. Synthesizing action sequences for modifying model decisions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5462–5469, 2020.

[67] Rawal, Kaivalya, Ece Kamar, and Himabindu Lakkaraju. Algorithmic recourse in the wild: Understanding the impact of data and model shifts. *arXiv preprint arXiv:2012.11788*, 2020.

[68] Ross, Alexis, Himabindu Lakkaraju, and Osbert Bastani. Learning models for actionable recourse. *Advances in Neural Information Processing Systems*, 34:18734–18746, 2021.

[69] Russell, Chris. Efficient search for diverse coherent explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 20–28, 2019.

[70] Shavit, Yonadav, Benjamin Edelman, and Brian Axelrod. Causal strategic linear regression. In *International Conference on Machine Learning*, pages 8676–8686. PMLR, 2020.

[71] Tolomei, Gabriele, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 465–474, 2017.

[72] Upadhyay, Sohini, Shalmali Joshi, and Himabindu Lakkaraju. Towards robust and reliable algorithmic recourse. *arXiv preprint arXiv:2102.13620*, 2021.

[73] Ustun, Berk, Alexander Spangher, and Yang Liu. Actionable recourse in linear classification. pages 10–19. doi: 10.1145/3287560.3287566.

[74] Veitch, Victor, Alexander D'Amour, Steve Yadlowsky, and Jacob Eisenstein. Counterfactual invariance to spurious correlations: Why and how to pass stress tests. *arXiv preprint arXiv:2106.00545*, 2021.

[75] Venkatasubramanian, Suresh and Mark Alfano. The philosophical basis of algorithmic recourse. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 284–293, 2020.

[76] Verma, Sahil, John Dickerson, and Keegan Hines. Counterfactual explanations for machine learning: A review, 2020.

[77] Verma, Sahil, Keegan Hines, and John P Dickerson. Amortized generation of sequential algorithmic recourses for black-box models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8512–8519, 2022.

[78] Kügelgen, Juliusvon , Amir-Hossein Karimi, Umang Bhatt, Isabel Valera, Adrian Weller, and Bernhard Schölkopf. On the fairness of causal algorithmic recourse. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9584–9594, 2022.

[79] Vos, Daniël and Sicco Verwer. Efficient training of robust decision trees against adversarial examples. In Meila, Marina and Tong Zhang, editors, *ICML*, 2021.

[80] Wang, Zijie J, Jennifer Wortman Vaughan, Rich Caruana, and Duen Horng Chau. Gam coach: Towards interactive and user-centered algorithmic recourse. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2023.

[81] Wei, Dennis. Decision-making under selective labels: Optimal finite-domain policies and beyond. In *International Conference on Machine Learning*, pages 11035–11046. PMLR, 2021.

[82] Wilhelm, Colin. Big data and the credit gap. https://www.politico.com/agenda/story/2018/02/07/big-data-credit-gap-000630, February 2018.

[83] Wykstra, S. Government's use of algorithm serves up false fraud charges. undark, 6 january, 2020.

[84] Yadav, Prateek, Peter Hase, and Mohit Bansal. Inspire: A framework for integrating individual user preferences in recourse.

[85] Yadav, Prateek, Peter Hase, and Mohit Bansal. Low-cost algorithmic recourse for users with uncertain cost functions. *arXiv preprint arXiv:2111.01235*, 2021.

A Notation

Symbol	Meaning
$\mathcal{X} \subseteq \mathbb{R}^d$	feature space
$\mathcal{Y} = \{-1, +1\}$	label space
$\mathcal{X}_j \subseteq \mathbb{R}$	feature space for feature j
$\boldsymbol{a} \in A(\boldsymbol{x})$	action vector from $\boldsymbol{x} \in \mathcal{X}$
$A(\boldsymbol{x})$	action set for $\boldsymbol{x} \in \mathcal{X}$. $\mathbf{0} \in A(\boldsymbol{x})$
$A_j(\boldsymbol{x})$	set of feasible actions for feature j from $\boldsymbol{x}$
$R_A(\boldsymbol{x}) := \{\boldsymbol{x} + \boldsymbol{a} \mid \boldsymbol{a} \in A(\boldsymbol{x})\}$	reachable set from $\boldsymbol{x}$
$R_A^{\text{int}}(\boldsymbol{x}) \subset R_A(\boldsymbol{x})$	inner approximation of a reachable set from $\boldsymbol{x}$
$N_A(\boldsymbol{x})$	set of sibling points from $\boldsymbol{x}$
$f : \mathcal{X} \to \mathcal{Y}$	classification model
$S^+ := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n^+}$ s.t. $y_i = +1$	a set of positive examples
$S^- := \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n^+}$ s.t. $y_i = -1$	a set of negative examples
$n^+ := \lvert S^+ \rvert$	number of positive examples in a sample
$n^- := \lvert S^- \rvert$	number of negative examples in a sample
$[k] := \{1, \ldots, k\}$	set of positive integers from 1 to k

Table 4: Table of Notation

B Proofs

In this Appendix, we present proofs of our claims in Section 3.

B.1 Proof of Theorem 7

Theorem 7. *Suppose we have a dataset of labeled examples $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Every model $f : \mathcal{X} \to \mathcal{Y}$ can provide recourse to $\boldsymbol{x}$ if:*

$$\mathsf{FNR}(f) < \frac{1}{n^+} \sum_{i=1}^{n} \mathbb{1}[\boldsymbol{x}_i \in R \ \wedge \ y_i = +1] \tag{4}$$

where $\mathsf{FNR}(f) := \frac{1}{n^+} \sum_{i=1}^n \mathbb{1}[f(\boldsymbol{x}_i) = -1 \ \wedge \ y_i = +1]$ is the false negative rate of f on $\mathcal{D}$ and where n^+ is number of positive examples in $\mathcal{D}$, and $R \subseteq R_A(\boldsymbol{x})$ is any subset of the reachable set.

Proof. The proof is based on an application of the pigeonhole principle over the positive examples $S^+ := \{\boldsymbol{x}_i \mid y_i = +1, i \in [n]\}$. Given a classifier f, denote the total number of true positive and false negative predictions over S^+ as:

$$\mathsf{TP}(f) := \sum_{i=1}^{n} \mathbb{1}[f(\boldsymbol{x}_i) = +1 \wedge y_i = +1] \qquad \mathsf{FN}(f) := \sum_{i=1}^{n} \mathbb{1}[f(\boldsymbol{x}_i) = -1 \wedge y_i = +1].$$

Suppose that for a given $R \subseteq R_A(\boldsymbol{x})$, the classifier satisfies the following condition:

$$\mathsf{TP}(f) > n^+ - |S^+ \cap R|.$$

In other words, the number of correct positive predictions exceeds the number of positive examples outside R. In this case, by the pigeonhole principle, the classifier f must assign a correct prediction to at least one of the positive examples in R – i.e., there exists a point $\boldsymbol{x}' \in S^+ \cap R$ such that $f(\boldsymbol{x}') = y_i = +1$. Given $R \subseteq R_A(\boldsymbol{x})$, we have that $\boldsymbol{x} \in R_A(\boldsymbol{x})$. Thus, we can reach $\boldsymbol{x}'$ from $\boldsymbol{x}$ by performing the action $\boldsymbol{a} = \boldsymbol{x}' - \boldsymbol{x}$ – i.e., we can change the prediction from $f(\boldsymbol{x}) = -1$ to $f(\boldsymbol{x} + \boldsymbol{a}) = +1$.

We recover the condition in the statement of the Theorem as follows:

$$\mathsf{TP}(f) > n^+ - |S^+ \cap R| \tag{5}$$

$$\mathsf{FN}(f) < |S^+ \cap R|, \tag{6}$$

$$\mathsf{FNR}(f) < \frac{1}{n^+} \sum_{i=1}^{n} \mathbb{1}[\boldsymbol{x}_i \in R \ \wedge \ y_i = +1] \tag{7}$$

Here, we proceed from Eqn. (5) to Eqn. (6) by using the fact that $\mathsf{TP}(f) = n^+ - \mathsf{FN}(f)$, and from Eqn. (6) to (7) by dividing both sides by $\frac{1}{n^+}$ and applying the definition of the false negative rate. □

B.2 Proof of Proposition 5

Proposition 5. *Any classification task with bounded features whose actions obey monotonicity constraints must contain at least one fixed point.*

Proof. Consider a set of d features $(x_1, \ldots, x_d) = \boldsymbol{x} \in \mathcal{X}$ over a bounded feature space. Let l_j and u_j denote the lower and upper bounds on feature j, so that $x_j \in [l_j, u_j]$ for all $j \in [d]$

We will proceed to construct a fixed point over $\mathcal{X}$ under the following conditions: (i) each feature is monotonically increasing, so that $a_j \geq 0$ for all $j \in [d]$; (ii) each feature is monotonically decreasing, so that $a_j \leq 0$ for all $j \in [d]$; (iii) each feature is either monotonically increasing or monotonically decreasing so that $a_j \geq 0$ or $a_j \leq 0$ for all $j \in [d]$.

In the case of (i), the fixed point corresponds to a feature vector $\boldsymbol{x} \in \mathcal{X}$ such that $x_j = u_j$ for all $j \in [d]$. We proceed by contradiction. Suppose $\boldsymbol{x}$ is not a fixed point, then there exists an action $\boldsymbol{a}' \in A(\boldsymbol{x})$ such that $\boldsymbol{a}' \neq \{\mathbf{0}\}$. In turn, there exists $a'_j > 0$. Let $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{a}'$ then $x'_j = x_j + a'_j > u_j$ which violates our initial assumption that $x'_j \in [l_j, u_j]$. Thus, $\boldsymbol{x}$ must be a fixed point.

In the case of (ii), the fixed point corresponds to a feature vector $\boldsymbol{x} \in \mathcal{X}$ such that $x_j = l_j$ for all $j \in [d]$. We proceed by contradiction. Suppose $\boldsymbol{x}$ is not a fixed point, then there exists an action $\boldsymbol{a}' \in A(\boldsymbol{x})$ such that $\boldsymbol{a}' \neq \{\mathbf{0}\}$. In turn, there exists $a'_j < 0$. Let $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{a}'$ then $x'_j = x_j + a'_j < l_j$ which violates our initial assumption that $x'_j \in [l_j, u_j]$. Thus, $\boldsymbol{x}$ must be a fixed point.

In the case of (iii), by combining the above, it can be seen that as long as x_j satisfies monotonicity constraints which can either be increasing or decreasing there must contain at least one fixed point.

$$x_j = \begin{cases} u_j, \text{if } j \in J_+ \\ l_j, \text{if } j \in J_- \end{cases}$$

where J_+ is the set of indices with monotonically increasing constraints and J_- is the set of indices with monotonically decreasing constraints.

□

B.3 Proof of Proposition 6

Proposition 6. *Consider adding a new feature $\mathcal{X}_{d+1} \subseteq \mathbb{R}$ to a set of d features $\mathcal{X} \subseteq \mathbb{R}^d$. Any fixed point $\boldsymbol{x} \in \mathcal{X}$ induces the following confined regions in the $(d+1)$-dimensional space:*

- *$|\mathcal{X}_{d+1}|$ fixed points in the $(d+1)$-dimensional feature space, if x_{d+1} is immutable.*
- *A fixed point $\boldsymbol{z}_0 := (\boldsymbol{x}, x_{d+1})$ where x_{d+1} is an* extreme point *of $\mathcal{X}_{d+1}$, that is, $x_{d+1} := \max \mathcal{X}_{d+1}$ or $x_{d+1} := \min \mathcal{X}_{d+1}$ if $(d+1)$-th feature is monotonically increasing (respectively, decreasing) in the action set $A(\boldsymbol{z}_0)$, and the constraints in $A(\boldsymbol{z}_0)$ are separable.*
- *A fixed region if $R_A(x_1, x_2, \ldots, x_d, x_{d+1}) = R_A(x_1, x_2, \ldots, x_d, x'_{d+1})$ for any two $x_{d+1}, x'_{d+1} \in \mathcal{X}_{d+1}$.*

Proof. Let us denote the $(d+1)$-dimensional feature space as $\bar{\mathcal{X}} := \mathcal{X}_1 \times \ldots \times \mathcal{X}_d \times \mathcal{X}_{d+1}$.

- Suppose a point $\boldsymbol{x}' \in \bar{\mathcal{X}}$ has the same feature values as $\boldsymbol{x}$ in its first d dimensions. As x_{d+1} is immutable, the only feasible action for x'_{d+1} is $a_{d+1} = 0$. This holds for any possible value of x'_{d+1}. This implies that for all feature values of the $(d+1)$-th feature, $\boldsymbol{x}'$ remains a fixed point. Therefore, there must exist $|\mathcal{X}_{d+1}|$ fixed points.
- Observe that if v_{d+1} is an extreme point, then the only possible action is $a_{d+1} = 0$ because the $d+1$-th feature must satisfy a monotonicity constraint. As the constraints in $A(\boldsymbol{z}_0)$ are separable by assumption, and $A(\boldsymbol{x}) = \{\mathbf{0}\}$, $\boldsymbol{z}_0$ must also have only one possible action $A(\boldsymbol{z}_0) = \{\mathbf{0}\}$.

- Given any $\boldsymbol{x}' \in \bar{\mathcal{X}}$ where the first d dimensions are the same as in $\boldsymbol{x}$, we have $R_A(\boldsymbol{x}') = R_A(\boldsymbol{x})$. As any other $\boldsymbol{x}'' \in R_A(\boldsymbol{x}')$ also shares the first d dimensions and is also $\boldsymbol{x}'' \in R_A(\boldsymbol{x})$, we have that $R_A(\boldsymbol{x}') \subseteq R_A(\boldsymbol{x})$.

□

C Supplement to Section 4 – Algorithms

In this Appendix, we describe how to formulate and solve the optimization problems in Section 4 as mixed-integer programs. We start by presenting a MIP formulation for the optimization problem solved in the FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ routine. We then describe how this formulation can be extended to an optimization problem in the IsReachable$(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x}))$ routine. Finally, we describe how this formulation can be extended to the complex actionability constraints in Table 1.

C.1 MIP Formulation for FindAction

Given a point $\boldsymbol{x} \in \mathcal{X}$, an action set $A(\boldsymbol{x})$, and a set of previous optima $\mathcal{A}^{\text{opt}}$, we can formulate FindAction$(\boldsymbol{x}, A(\boldsymbol{x}))$ as the following mixed-integer program:

$$\min_{\boldsymbol{a}} \quad \sum_{j \in [d]} a_j^+ + a_j^-$$

$$\begin{aligned}
\text{s.t.} \quad & a_j^+ \geq a_j && j \in [d] && \textit{positive component of } a_j && \text{(8a)}\\
& a_j^- \geq -a_j && j \in [d] && \textit{negative component of } a_j && \text{(8b)}\\
& a_j = a_{j,k} + \delta_{j,k}^+ - \delta_{j,k}^- && j \in [d], \boldsymbol{a}_k \in \mathcal{A}^{\text{opt}} && \textit{distance from prior actions} && \text{(8c)}\\
& \varepsilon_{\text{min}} \leq \sum_{j \in [d]} (\delta_{j,k}^+ - \delta_{j,k}^-) && \boldsymbol{a}_k \in \mathcal{A}^{\text{opt}} && \textit{any solution is } \varepsilon_{min} \textit{ away from } \boldsymbol{a}_k && \text{(8d)}\\
& \delta_{j,k}^+ \leq M_{j,k}^+ u_{j,k} && j \in [d], \boldsymbol{a}_k \in \mathcal{A}^{\text{opt}} && \delta_{j,k}^+ > 0 \implies u_{j,k} = 1 && \text{(8e)}\\
& \delta_{j,k}^- \leq M_{j,k}^- (1 - u_{j,k}) && j \in [d], \boldsymbol{a}_k \in \mathcal{A}^{\text{opt}} && \delta_{j,k}^- > 0 \implies u_{j,k} = 0 && \text{(8f)}\\
& a_j \in A_j(\boldsymbol{x}) && j \in [d] && \textit{separable actionability constraints on } j && \text{(8g)}\\
& \delta_{j,k}^+, \delta_{j,k}^- \in \mathbb{R}_+ && j \in [d] && \textit{signed distances from } a_{j,k} && \text{(8h)}\\
& u_{j,k} \in \{0, 1\} && j \in [d] && u_{j,k} := \mathbb{1}[\delta_{j,k}^+ > 0] && \text{(8i)}
\end{aligned}$$

The formulation finds action in the set $\boldsymbol{a} \in A(\boldsymbol{x}) / \mathcal{A}^{\text{opt}}$ by combining two classes of constraints: (i) constraints to restrict actions $\boldsymbol{a} \in A(\boldsymbol{x})$ and (ii) constraints to rule out actions in $\boldsymbol{a} \in \mathcal{A}^{\text{opt}}$.

The formulation encodes the separable constraints in $A(\boldsymbol{x})$ – i.e., a constraint that can be enforced for each feature. The formulation must be extended with additional variables and constraints to handle constraints as discussed in Appendix C.3. These constraints are handled through the $a_j \in A_j(\boldsymbol{x})$ conditions in Constraint 8g. This constraint can handle a number of actionability constraints that can be passed solver when defining the variables a_j, including *bounds* (e.g., $a_j \in [-x_j, 10 - x_j]$), *integrality* (e.g., $a_j \in \{0, 1\}$ or $a_j \in \{L - x_j, L - x_j + 1, \ldots, U - x_j\}$), and *monotonicity* (e.g., $a_j \geq 0$ or $a_j \leq 0$).

The formulation rules out actions in $\boldsymbol{a} \in \mathcal{A}^{\text{opt}}$ through the "no good" constraints in Constraints (8c) to (8f). Here, Constraint (8d) ensures feasible actions from previous solutions by at least ε_{min}. We set to a sufficiently small number $\varepsilon_{\text{min}} := 10^-6$ by default, but use larger values when working with discrete feature sets (e.g., $\varepsilon_{\text{min}} = 1$ for cases where every actionable feature is binary or integer-valued). Constraints (8e) and (8f) ensure that either $\delta_{j,k}^+ > 0$ or $\delta_{j,k}^- > 0$. These are "Big-M constraints" where the Big-M parameters can be set to represent the largest value of signed distances. Given an action $a_j \in [a_j^{\text{LB}}, a_j^{\text{UB}}]$, we can set $M_{j,k}^+ := |a_j^{\text{UB}} - a_{j,k})$ and $M_{j,k}^- := |a_{j,k} - a_j^{\text{LB}}|$.

The formulation chooses each action in $\boldsymbol{a} \in A(\boldsymbol{x}) / \mathcal{A}^{\text{opt}}$ to minimize the L_1 norm. We compute the L_1-norm component-wise as $|a_j| := a_j^+ + a_j^-$ where the variables a_j^+ and a_j^- are set to the positive and negative

components of $|a_j|$ in Constraints (8a) and (8b). This choice of objective is meant to induce sparsity among the actions we recover by repeatedly solving Algorithm 1. Given that the objective function does not affect the feasibility of the optimization problem, one could set the objective to 1 when solving the problem for fixed-point detection.

C.2 MIP Formulation for IsReachable

Given a point $\boldsymbol{x} \in \mathcal{X}$, an action set $A(\boldsymbol{x})$, we can formulate the optimization problem for $\mathsf{IsReachable}(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x}))$ as a special case of the MIP in (8) in which we set $\mathcal{A}^{\text{opt}} = \emptyset$ and include the constraint $\boldsymbol{a} = \boldsymbol{x} - \boldsymbol{x}'$. In this case, any feasible solution would certify that $\boldsymbol{x}'$ can be attained from $\boldsymbol{x}$ using the actions in $A(\boldsymbol{x})$. Thus, we can return $\mathsf{IsReachable}(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x})) = 1$ if the MIP is feasible and $\mathsf{IsReachable}(\boldsymbol{x}, \boldsymbol{x}', A(\boldsymbol{x})) = 0$ if it is infeasible.

C.3 Encoding Actionability Constraints

We describe how to extend the MIP formulation in (8) to encode salient classes of actionability constraints. Our software includes an ActionSet API that allows practitioners to specify these constraints across each MIP formulation.

C.3.1 Encoding Preservation for Categorical Features

Many datasets contain subsets of features that reflect the underlying value of a categorical attribute. For example, a dataset may encode a categorical attribute with $K = 3$ categories such `marital_status` $\in \{single, married, other\}$ using a subset of $K - 1 = 2$ features such as `married` and `single`. In such cases, actions on these features must obey non-separable actionability constraints to preserve the encoding – i.e., to ensure that a person cannot be `married` and `single` at the same time.

We can enforce these conditions by adding the following constraints to the MIP Formulation in (8):

$$L \leq \sum_{j \in \mathcal{J}} x_j + a_j \leq U \tag{9}$$

Here, $\mathcal{J} \subseteq [d]$ is the index set of features with encoding constraints, and L and U are lower and upper limits on the number of features in $\mathcal{J}$ that must hold to preserve an encoding. Given a standard one-hot encoding of a categorical variable with K categories, $\mathcal{J}$ would contain the indices of $K - 1$ features (i.e., dummy variables for the $K - 1$ categories other than the reference category). We would ensure that all actions preserve this encoding by setting $L = 0$ and $U = 1$.

C.3.2 Logical Implications & Deterministic Causal Relationships

Datasets often include features where actions on one feature will induce changes in the values and actions for other features. For example, in Table 1, changing `is_employed` from `FALSE` to `TRUE` would change the value of `work_hrs_per_week` from 0 to a value ≥ 0.

We capture these conditions by adding variables and constraints that capture logical implications in action space. In the simplest case, these constraints would relate the values for a pair of features $j, j' \in [d]$ through an if-then condition such as: "if $a_j \geq v_j$ then $a'_j = v_{j'}$". In such cases, we could capture this relationship by

adding the following constraints to the MIP Formulation in (8):

$$Mu \geq a_j - v_j \tag{10}$$
$$M(1-u) \geq v_j - a_j \tag{11}$$
$$uv_{j'} = a_{j'} \tag{12}$$
$$u \in \{0,1\}$$

The constraints shown above capture the "if-then" condition by introducing a binary variable $u := \mathbb{1}[a_j \geq v_j]$. The indicator is set through the Constraints (10) and (11) where $M := a_j^{\text{UB}} - v_j$. If the implication is met, then $a_{j'}$ is set to $v_{j'}$ through Constraint (12). We apply this approach to encode a number of salient actionability constraints shown in Table 1 by generalizing the constraint shown above to a setting where: (i) the "if" and "then" conditions to handle subsets of features, and (ii) the implications link actions on mutable features to actions on an immutable feature (i.e. so that actions on a mutable feature `years_since_last_application` will induce changes in an immutable feature `age`).

C.3.3 Custom Reachability Conditions

We now describe a general-purpose solution to specify "reachable" values for a subset of discrete features. These constraints can be used when we need to encode constraints that require fine-grained control over the actionability of different features. For example, when specifying actions over one-hot encoding of ordinal features (e.g., `max_degree_BS` and `max_degree_MS` as in Table 1) or as 'thermometer encoding" (e.g.,`monthly_income_geq_2k`, `monthly_income_geq_5k`, `monthly_income_geq_10k`). In such cases, we can formulate a set of custom reachability constraints over these features given the following inputs:

- index set of features $\mathcal{J} \subset [d]$,
- V, a set of all valid values that can be realized by the features in $\mathcal{J}$.
- $E \in \{0,1\}^{k \times k}$, a matrix whose entries encode the reachability of points in V: $e_{i,j} = 1$ if and only if point v_i can reach point v_j for $v_i, v_j \in V$.

Given these inputs, we add the following constraints for each $j \in \mathcal{J}$ to the MIP Formulation in (8):

$$a_j = \sum_{k \in E[i]} e_{i,k} a_{j,k} u_{j,k} \tag{13}$$
$$1 = \sum_{k \in E[i]} u_{j,k} \tag{14}$$
$$u_{j,k} \leq e_{i,k} \tag{15}$$
$$u_{j,k} \in \{0,1\}$$

Here, $u_{j,k} := \mathbb{1}[\boldsymbol{x}' \in V]$ indicates if we choose an action to attain point $\boldsymbol{x}' \in V$. Constraint (13) defines the set of reachable points from i, while Constraint (13) ensures that only one such point can be selected. Here, $e_{i,k}$ is a parameter obtained from the entries of E for point i, and the values of $a_{j,k}$ are set as the differences from x_j to x'_j where $\boldsymbol{x}, \boldsymbol{x}' \in V$. We present examples of how to use these constraints to preserve a one-hot encoding over ordinal features in Fig. 3, and to preserve a thermometer encoding in Fig. 4.

	V		
IsEmployedLeq1Yr	IsEmployedBt1to4Yrs	IsEmployedGeq4Yrs	E
0	0	0	$[1, 1, 0, 0]$
1	0	0	$[0, 1, 1, 0]$
0	1	0	$[0, 0, 1, 1]$
0	0	1	$[0, 0, 0, 1]$

Figure 3: Here V denotes valid combinations of features in columns 1 - 3. E in column 4 and shows which points can be reached. For example, $[1, 1, 0, 0]$ represents point $[0, 0, 0]$ can be reached and point $[1, 0, 0]$ can be reached, but no other points can be reached.

	V		
NetFractionRevolvingBurdenGeq90	NetFractionRevolvingBurdenGeq60	NetFractionRevolvingBurdenLeq30	E
0	0	0	$[1, 1, 0, 0]$
1	0	0	$[0, 1, 0, 0]$
0	1	0	$[1, 1, 1, 0]$
0	1	1	$[1, 1, 1, 1]$

Figure 4: Here V denotes valid combinations of features in columns 1 - 3. For these features, we wanted to produce actions that would reduce `NetFractionRevolvingBurden` for consumers. E in column 4 and shows which points can be reached. For example, $[1, 1, 0, 0]$ represents point $[0, 0, 0]$ can be reached, and point $[1, 0, 0]$ can be reached, but no other points can be reached.

D Supplement to Section 5 – Experiments

For each dataset, the Simple action set contains only immutability features and integrality constraints. The Separable action set contains the same actionability constraints as simple and adds monotonicity constraints. The Non-Separable action set contains all the actionability constraints as separable and adds non-separable constraints.

D.1 Actionability Constraints for the `heloc` Dataset

We use the action set shown in Table 8. The non-separable constraints are described in Appendix E.

D.2 Actionability Constraints for the `givemecredit` Dataset

We show a list of all features and their separable actionability constraints in Table 5. The non-separable actionability constraints for this dataset include:

1. Logical Implications on `AnyRealEstateLoans` and `MultipleRealEstateLoans`. Here, if `AnyRealEstateLoans` changes from 1 to 0, then `MultipleRealEstateLoans` must also change from 1 to 0.

2. Logical Implications on `AnyOpenCreditLinesAndLoans` and `MultipleOpenCreditLinesAndLoans`. Here, if `AnyOpenCreditLinesAndLo` changes from 1 to 0, then `MultipleOpenCreditLinesAndLoans` must also change from 1 to 0.

3. Custom Constraints to Preserve Thresholds for features `MonthlyIncomeIn1000sGeq2`, `MonthlyIncomeIn1000sGeq5`, `MonthlyIncomeGeq7K`. An example can be found in Fig. 4. Here the feasible actions increase the consumer's `MonthlyIncome` and the maximum value a user can have is where `MonthlyIncomeGeq2K` = 1, `MonthlyIncomeGeq5K` = 1, and `MonthlyIncomeIn1000sGeq7` = 1

4. Custom Constraints to Preserve Thresholds for features `TotalCreditBalanceGeq1K`, `TotalCreditBalanceGeq2K`, `TotalCreditBalanceGeq5K`. An example can be found in figure 4. Here the feasible actions decrease the consumer's `TotalCreditBalance` and the minimum value a consumer can have is where `TotalCreditBalanceGeq1K` = 0, `TotalCreditBalanceGeq2K` = 0, and `TotalCreditBalanceGeq5K` = 0

Feature	LB	UB	Type	Actionable	Monotonicity
AvgYearsInFileGeq3	0.0	1.0	boolean	T	0
AvgYearsInFileGeq5	0.0	1.0	boolean	T	0
AvgYearsInFileGeq7	0.0	1.0	boolean	T	0
AvgYearsInFileGeq9	0.0	1.0	boolean	T	0
ExternalRiskEstimate	36.0	89.0	integer	F	0
InitialYearsOfAcctHistory	0.0	2.0	integer	T	+
ExtraYearsOfAcctHistory	0.0	48.0	integer	F	0
MostRecentTradeWithinLastYear	0.0	1.0	boolean	T	+
MostRecentTradeWithinLast2Years	0.0	1.0	boolean	T	+
AnyDerogatoryComment	0.0	1.0	boolean	F	0
AnyDelTradeInLastYear	0.0	1.0	boolean	F	0
AnyTrade120DaysDelq	0.0	1.0	boolean	F	0
AnyTrade90DaysDelq	0.0	1.0	boolean	F	0
AnyTrade60DaysDelq	0.0	1.0	boolean	F	0
AnyTrade30DaysDelq	0.0	1.0	boolean	F	0
NumInstallTrades	0.0	55.0	integer	F	0
NumInstallTradesWBalance	1.0	23.0	integer	F	0
NumRevolvingTrades	1.0	85.0	integer	F	0
NumRevolvingTradesWBalance	0.0	32.0	integer	T	-
NetFractionInstallBurdenGeq90	0.0	1.0	boolean	F	0
NetFractionInstallBurdenGeq70	0.0	1.0	boolean	F	0
NetFractionInstallBurdenGeq50	0.0	1.0	boolean	F	0
NetFractionInstallBurdenGeq30	0.0	1.0	boolean	F	0
NetFractionInstallBurdenGeq10	0.0	1.0	boolean	F	0
NetFractionInstallBurdenEq0	0.0	1.0	boolean	F	0
NetFractionRevolvingBurdenGeq90	0.0	1.0	boolean	T	0
NetFractionRevolvingBurdenGeq60	0.0	1.0	boolean	T	0
NetFractionRevolvingBurdenLeq30	0.0	1.0	boolean	T	0
NumBank2NatlTradesWHighUtilizationGeq2	0.0	1.0	boolean	T	-

Table 5: Overview of Separable Actionability Constraints for the `givemecredit` dataset.

D.3 Actionability Constraints for the `german` Dataset

We show a list of all features and their separable actionability constraints in Table 6. The non-separable actionability constraints for this dataset include:

1. One Hot Encoding for features `savings_acct_le_100`, `savings_acct_bt_100_499`, `savings_acct_bt_500_999`, `savings_acct_ge_1000` An example of this can be found in Appendix C.3.1. Here, actions must restrict only one category to be selected.

Feature	LB	UB	Actionable	Monotonicity
age	19	75	F	
is_male	0	1	F	
is_foreign_worker	0	1	F	
has_liable_persons	1	1	F	
max_approved_loan_duration_geq_10_m	0	1	F	
max_approved_loan_amt_geq_10k	0	1	F	
max_approved_loan_rate_geq_2	0	1	F	
credit_history_no_credits_taken	0	1	F	
credit_history_all_credits_paid_till_now	0	1	F	
credit_history_delay_or_critical_in_payment	0	1	F	
loan_required_for_car	0	1	F	
loan_required_for_home	0	1	F	
loan_required_for_education	0	1	F	
loan_required_for_business	0	1	F	
loan_required_for_other	0	1	F	
max_val_checking_acct_ge_0	0	1	T	+
max_val_savings_acct_ge_0	0	1	T	+
years_at_current_home_ge_2	0	1	T	+
employed_ge_4_yr	0	1	T	+
savings_acct_le_100	0	1	T	0
savings_acct_bt_100_499	0	1	T	0
savings_acct_bt_500_999	0	1	T	0
savings_acct_ge_1000	0	1	T	0
has_history_of_installments	0	1	T	+

Table 6: Overview of Separable Actionability Constraints for the `german` dataset.

D.4 Overview of Model Performance

Dataset	Model Type	Sample	AUC
heloc	LR	Train	0.738
heloc	LR	Test	0.730
heloc	XGB	Train	0.733
heloc	XGB	Test	0.737
givemecredit	LR	Training	0.653
givemecredit	LR	Test	0.644
givemecredit	XGB	Training	0.651
givemecredit	XGB	Test	0.640
german	LR	Training	0.752
german	LR	Test	0.690
german	XGB	Training	0.753
german	XGB	Test	0.690

Table 7: Performance of LR and XGB models for all 3 datasets. We show the performance of each model on the training dataset and a held-out dataset. We perform a random grid search to tune the hyperparameters for each model and split the train and test by 80%/ 20%. We use the entire dataset to calculate the number of predictions without recourse.

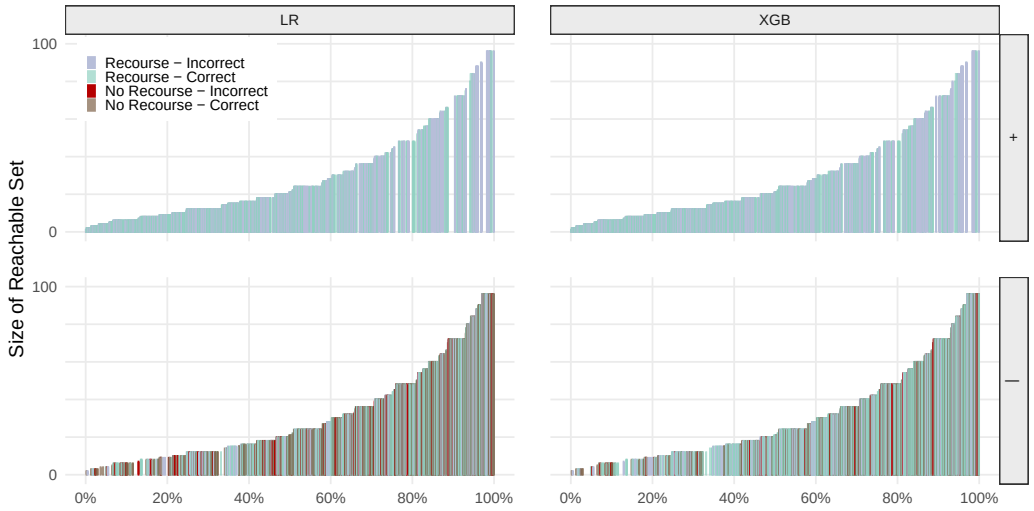

Figure 5: Composition of reachable sets for the `heloc` for LR and XGB. Each plot shows the size of reachable sets for each training example delineated by Algorithm 1. The top row displays sizes of reachable sets for samples with negative predictions and the bottom row for samples with positive predictions. Correct/incorrect denotes where the true label does/does not equal the predicted label and Recourse/No Recourse denotes if recourse is feasible/infeasible. We highlight predictions without recourse for both correctly classified and incorrectly classified negative points. We can see predictions without recourse are prevalent with all reachable set sizes and can be drastically different between classifiers.

E Demonstrations

E.1 Ensuring Recourse in Lending

Actionability Constraints We show a list of all features and their separable actionability constraints in Table 8. The non-separable actionability constraints for this dataset include:

1. Logical Implications on `MostRecentTradeInLastYear` and `MostRecentTradeInLast2Years` is explained in section Appendix C.3.2. Here, if `MostRecentTradeInLastYear` changes from 0 to 1 then `MostRecentTradeInLast2Years` must also change from 0 to 1.

2. Custom Constraints to Preserve Thresholds for features `NetFractionRevolvingBurdenGeq90`, `NetFractionRevolvingBurdenGeq60` `NetFractionRevolvingBurdenLeq30`. An example can be found in figure 4. Here, feasible actions must decrease the consumer's `NetFractionRevolvingBurden`. Therefore, the lowest category a consumer can reach is `NetFractionRevolvingBurdenLeq30` = 1.

Additional Results

Feature	LB	UB	Actionable	Monotonicity
AvgYearsInFileGeq3	0	1	T	0
AvgYearsInFileGeq5	0	1	T	0
AvgYearsInFileGeq7	0	1	T	0
AvgYearsInFileGeq9	0	1	T	0
ExternalRiskEstimate	36	89	F	
InitialYearsOfAcctHistory	0	2	T	+
ExtraYearsOfAcctHistory	0	48	F	
MostRecentTradeWithinLastYear	0	1	T	+
MostRecentTradeWithinLast2Years	0	1	T	+
AnyDerogatoryComment	0	1	F	
AnyDelTradeInLastYear	0	1	F	
AnyTrade120DaysDelq	0	1	F	
AnyTrade90DaysDelq	0	1	F	
AnyTrade60DaysDelq	0	1	F	
AnyTrade30DaysDelq	0	1	F	
NumInstallTrades	0	55	F	
NumInstallTradesWBalance	1	23	F	
NumRevolvingTrades	1	85	F	
NumRevolvingTradesWBalance	0	32	T	-
NetFractionInstallBurdenGeq90	0	1	F	
NetFractionInstallBurdenGeq70	0	1	F	
NetFractionInstallBurdenGeq50	0	1	F	
NetFractionInstallBurdenGeq30	0	1	F	
NetFractionInstallBurdenGeq10	0	1	F	
NetFractionInstallBurdenEq0	0	1	F	
NetFractionRevolvingBurdenGeq90	0	1	T	0
NetFractionRevolvingBurdenGeq60	0	1	T	0
NetFractionRevolvingBurdenLeq30	0	1	T	0
NumBank2NatlTradesWHighUtilizationGeq2	0	1	T	-

Table 8: Overview of Separable Actionability Constraints for the heloc dataset.

		Actions						
Feature	$\boldsymbol{x}$	$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$\boldsymbol{a}_3$	$\boldsymbol{a}_4$	$\boldsymbol{a}_5$	$\boldsymbol{a}_6$	$\boldsymbol{a}_7$
AvgYearsInFileGeq3	1	-	-	-	-	-	-	-
AvgYearsInFileGeq5	0	1	-	1	1	1	1	1
AvgYearsInFileGeq7	0	-	-	1	-	1	1	1
AvgYearsInFileGeq9	0	-	-	-	-	-	1	1
ExternalRiskEstimate	59	-	-	-	-	-	-	-
InitialYearsOfAcctHistory	2	-	-	-	-	-	-	-
ExtraYearsOfAcctHistory	8	-	-	-	-	-	-	-
MostRecentTradeWithinLastYear	1	-	-	-	-	-	-	-
MostRecentTradeWithinLast2Years	1	-	-	-	-	-	-	-
AnyDerogatoryComment	0	-	-	-	-	-	-	-
AnyDelTradeInLastYear	1	-	-	-	-	-	-	-
AnyTrade120DaysDelq	0	-	-	-	-	-	-	-
AnyTrade90DaysDelq	0	-	-	-	-	-	-	-
AnyTrade60DaysDelq	1	-	-	-	-	-	-	-
AnyTrade30DaysDelq	0	-	-	-	-	-	-	-
NumInstallTrades	8	-	-	-	-	-	-	-
NumInstallTradesWBalance	2	-	-	-	-	-	-	-
NumRevolvingTrades	7	-	-	-	-	-	-	-
NumRevolvingTradesWBalance	1	-	-1	-	-1	-1	-	-1
NetFractionInstallBurdenGeq90	0	-	-	-	-	-	-	-
NetFractionInstallBurdenGeq70	1	-	-	-	-	-	-	-
NetFractionInstallBurdenGeq50	1	-	-	-	-	-	-	-
NetFractionInstallBurdenGeq30	1	-	-	-	-	-	-	-
NetFractionInstallBurdenGeq10	1	-	-	-	-	-	-	-
NetFractionInstallBurdenEq0	0	-	-	-	-	-	-	-
NetFractionRevolvingBurdenGeq90	0	-	-	-	-	-	-	-
NetFractionRevolvingBurdenGeq60	0	-	-	-	-	-	-	-
NetFractionRevolvingBurdenLeq30	1	-	-	-	-	-	-	-
NumBank2NatlTradesWHighUtilizationGeq2	0	-	-	-	-	-	-	-

Table 9: The prototype discussed in Section 6. We choose to highlight this example as the consumer has negative classifications for both LR and XGB for all possible actions. Although this consumer has feasible actions they are still unable to obtain recourse since every reachable point is negatively classified. In this demo, there are 453 examples of consumers that may have feasible actions, but they are still predictions without recourse by LR and XGB. In this table, $\boldsymbol{x}$ represents all the feature values for this consumer. Actions $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_7$ represent all the feasible actions for this consumer.

Feature \ ID	1064	2539	1403	1447	1822
AvgYearsInFileGeq3	1	1	1	1	1
AvgYearsInFileGeq5	1	1	1	1	1
AvgYearsInFileGeq7	1	1	1	1	1
AvgYearsInFileGeq9	1	1	1	1	1
ExternalRiskEstimate	66.0	67.0	76.0	86.0	86.0
InitialYearsOfAcctHistory	2.0	2.0	2.0	2.0	2.0
ExtraYearsOfAcctHistory	14.0	25.0	20.0	16.0	18.0
MostRecentTradeWithinLastYear	1	1	1	1	1
MostRecentTradeWithinLast2Years	1	1	1	1	1
AnyDerogatoryComment	1	1	1	0	0
AnyDelTradeInLastYear	1	0	0	0	0
AnyTrade120DaysDelq	0	0	0	0	0
AnyTrade90DaysDelq	0	0	0	0	0
AnyTrade60DaysDelq	0	0	0	0	0
AnyTrade30DaysDelq	0	0	0	1	1
NumInstallTrades	4.0	13.0	6.0	6.0	8.0
NumInstallTradesWBalance	3.0	4.0	2.0	3.0	3.0
NumRevolvingTrades	7.0	9.0	10.0	12.0	15.0
NumRevolvingTradesWBalance	0	0	0	0	0
NetFractionInstallBurdenGeq90	0	0	0	0	0
NetFractionInstallBurdenGeq70	0	1	0	1	0
NetFractionInstallBurdenGeq50	1	1	1	1	0
NetFractionInstallBurdenGeq30	1	1	1	1	1
NetFractionInstallBurdenGeq10	1	1	1	1	1
NetFractionInstallBurdenEq0	0	0	0	0	0
NetFractionRevolvingBurdenGeq90	0	0	0	0	0
NetFractionRevolvingBurdenGeq60	0	0	0	0	0
NetFractionRevolvingBurdenLeq30	1	1	1	1	1
NumBank2NatlTradesWHighUtilizationGeq2	0	0	0	0	0
Ground truth	-	-	+	+	-
Prediction (LR)	53.00%	60.00%	78.00%	87.00%	91.00%
Prediction (XGBoost)	+	+	+	+	+

Table 10: Fixed Points for the demonstration in Section 6.

E.2 Certifying Adversarial Robustness in Content Moderation

Our machinery can also certify adversarial robustness to manipulations that normally cannot be captured by traditional threat models such as perturbations within an L_p ball [see, e.g., 33, 55, 39, 79, 9]. In this demonstration, we show that our methods enable to reason about the behavior of arbitrary models under semantically meaningful adversarial manipulations of the feature vectors. Specifically, we do so by building action sets that encode constraints from Table 1. In what follows, we showcase this by evaluating the adversarial robustness of a bot detector on a social media platform.

Feature	LB	UB	Actionable
source_automation	0	1	F
source_other	0	1	F
source_branding	0	1	F
source_mobile	0	1	F
source_web	0	1	F
source_app	0	1	F
follower_friend_ratio	0.83	1.16×10^5	F
age_of_account_in_days_geq_365	0	1	T
age_of_account_in_days_geq_730	0	1	T
age_of_account_in_days_le_365	0	1	T
user_replied_geq_10	0	1	T
user_replied_geq_100	0	1	T
user_replied_le_10	0	1	T
user_favourited_geq_1000	0	1	T
user_favourited_geq_10000	0	1	T
user_favourited_le_1000	0	1	T
user_retweeted_geq_1	0	1	T
user_retweeted_geq_10	0	1	T
user_retweeted_geq_100	0	1	T
user_retweeted_le_1	0	1	T

Table 11: Features used for the Twitter bot detector. The groups of features `age_of_account_*`, `user_replied_*`, `user_favourited_*`, and `user_retweeted_*` are non-separable thermometer-encoded.

Setup We use the dataset of Twitter accounts from April 2016 annotated by experts [23] as genuine ("human") labeled as $y = +1$ or those representing inauthentic behavior ("bot") labeled as $y = -1$. As before, we consider a processed version with $n = 1\,438$ accounts and $d = 20$ features on their account history and activity (e.g., age of account, number of tweets, re-tweets, replies, use of apps), listed in Table 11. As in Section 6, we train a logistic regression and an XGBoost model. We set aside 287 accounts (20%) as a held-out test dataset.

Our goal is to demonstrate the use of Algorithm 2 for evaluating the robustness of a detector to adversarial manipulations. We assume that the adversary starts with a bot account that is correctly detected as bot, and aims to modify the features of the account until it is classified as human. The capabilities of the adversary include procuring additional tweets, retweets, and replies; waiting to increase the account age, and adding tweets from previously unused categories of apps. As this is a complex model of adversarial

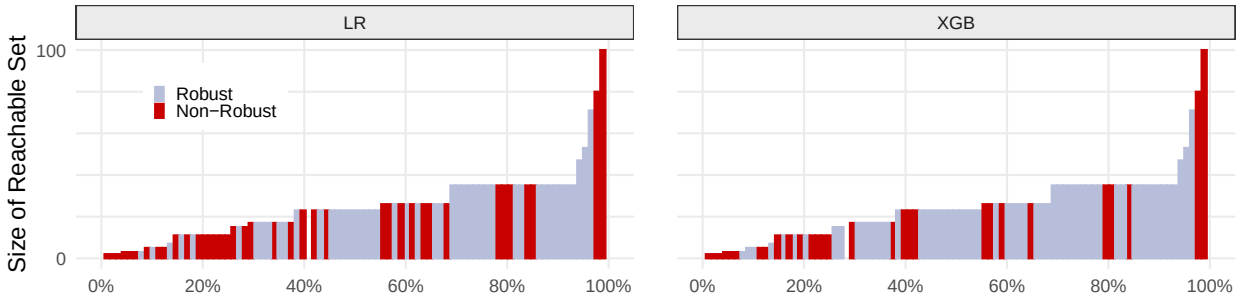

Figure 6: Composition of reachable sets for Twitter bot detection with LR and XGB models. Each plot shows the size of reachable sets generated by Algorithm 1 for every correctly classified bot account in the test set. Robust/Non-Robust denotes if the bot example can flip the prediction via manipulations within action set or not.

Model Type	AUC	Error	Robust Error
LR	0.697	34.1%	44.8%
XGB	0.698	34.5%	**33.3%**

Table 12: Robust error and performance of LR and XGB models trained for Twitter inauthentic behavior detection task. All metrics are computed on the test data.

capabilities which includes non-separable constraints, it cannot be captured by the commonly considered box constraints or L_p distances.

To evaluate adversarial robustness, we perform the following procedure. We run Algorithm 2 to generate reachable sets for all correctly classified bot accounts. We then evaluate the prediction of the detector on each of the points in the corresponding reachable set. Second, we measure adversarial robustness through a version of the *robust error* metric [as per 45]: the proportion of the bot accounts from the test set that are correctly classified as bots yet can have their predictions altered through adversarial actions. Formally, for a set of correctly predicted bot examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ from the test data, i.e., such that every $y_i = -1$ ("bot") and $f(\boldsymbol{x}_i) = -1$, we define the robust error as:

$$\frac{1}{m}\sum_{i=1}^{m} \mathbb{1}[\exists \boldsymbol{x}' \in R_A(\boldsymbol{x}_i) \;\text{ s.t. }\; f(\boldsymbol{x}') = +1]. \tag{16}$$

Results In our test data, we have 88 (out of 287 total accounts) bot accounts that are correctly classified as bots. We generate the 88 corresponding reachable sets for each account, and evaluate the predictions in each. Fig. 6 shows the distribution of reachable set sizes.

To evaluate the robustness of classifiers, in Table 12, we show the performance metrics of the classifiers along with the computed robust error. We find that for the majority of bots it is not possible to flip their prediction with any possible action within the adversarial model, with the robust error being approximately 33.3% for XGB and 44.82% for LR. Despite both classifier attaining similar error and AUC, XGB is more robust to adversarial manipulations.

In summary, our method enables to find adversarial examples, thus evaluate adversarial robustness, in tabular domains under a complex model of adversarial capabilities.

					Actionability Constraints Supported					
Recourse Method	**Reference**	**Certificate Of Infeasibility**	**Model Class**	**Back-End Method**	**Immutability**	**Interval**	**Monotonicity**	**Categorical Feature Encoding**	**Ordinal Feature Encoding**	**Logical Implications**
Efficient Search	Russell [69]	✓	linear	MIP	✓	✓	✓	✓	∗	∗
Actionable Recourse	Ustun et al. [73]	✓	linear	MIP	✓	✓	✓	∗	∗	∗
Probabilistic Recourse	Karimi et al. [41]	✗	differentiable	Gradient Based	✓	✓	✗	✗	✗	✓
Fair Causal Recourse	von Kügelgen et al. [78]	✗	differentiable	Gradient Based	✓	✓	✗	✗	✗	✓
DiCE	Mothilal et al. [58]	✗	differentiable	Gradient	✓	✓	✓	✓	✓	✗
FACE	Poyiadzi et al. [63]	✗	differentiable	Heuristic	✓	∗	✓	✗	✗	✗
DECE	Cheng et al. [11]	✗	differentiable	Gradient Based	✓	✓	✓	✗	✗	✗
OAE	Cui et al. [13]	✓	rule based	ILP	∗	∗	∗	∗	✗	✗
MACE	Karimi et al. [40]	✓	MLP, decision trees, RF	SAT	✓	✓	∗	✓	✓	∗
Reach		✓	model agnostic	MIP	✓	✓	✓	✓	✓	✓

Table 13: This table is a breakdown of the actionability constraints in Table 1 that are supported by recourse-provision methods in the prior literature. ✓ denotes that the constraint is supported in theory and implemented. ∗ denotes that the method could support the constraint in theory but it is not implemented. ✗ denotes that actionability constraint is not supported and was not discussed in the original paper. Note of the methods supports all of the actionability constraints in Table 1. In practice, this leads to loopholes and blindspots defined in Section 2.

F Additional Figures

```
import pandas as pd
import dice_ml
from sklearn.linear_model import LogisticRegression


X_names = ['years_employed', 'age']
y_name = 'repayment'

df = pd.DataFrame(
      columns = X_names + [y_name],
      data = [[2, 25, 0]] * 20
           + [[2, 25, 1]] * 2
           + [[4, 27, 0]] * 10
           + [[4, 27, 1]] * 20
           + [[10, 37, 0]] * 5
           + [[10, 37, 1]] * 20
      )

clf = LogisticRegression().fit(X = df[X_names], y = df[y_name])

explainer = dice.Dice(
      data_interface = dice_ml.Data(
      dataframe=df,
      continuous_features=X_names,
      outcome_name=y_name
      ),
      model_interface = dice_ml.Model(model=clf, backend='sklearn')
      )

# generates reachable points for years_employed = 2 and age = 25 which is negatively predicted for repayment
query_instance = df.loc[(df[X_names[0]] == 2) & (df[X_names[1]] == 25)].drop(columns=y_name).drop_duplicates()
explanation = explainer.generate_counterfactuals(
         query_instance,
         total_CFs=4,
         permitted_range={X_names[0]: [2, 30], X_names[1]: [25, 50]},
         desired_class="opposite",
         random_seed=201
         )

explanation.visualize_as_dataframe()
```

Listing 1: Illustration of a loophole that occurs with DiCE. The example dataset has two features: `years_employed` and `age`. All the reachable points returned by DiCE violate the determistic casuality constraint (see Table 1), thus creating loopholes.