Explanations for Monotonic Classifiers

Joao Marques-Silva [1] **Thomas Gerspacher** [1] **Martin Cooper** [1] **Alexey Ignatiev** [2] **Nina Narodytska** [3]

Abstract

In many classification tasks there is a requirement of monotonicity. Concretely, if all else remains constant, increasing (resp. decreasing) the value of one or more features must not decrease (resp. increase) the value of the prediction. Despite comprehensive efforts on learning monotonic classifiers, dedicated approaches for explaining monotonic classifiers are scarce and classifier-specific. This paper describes novel algorithms for the computation of one formal explanation of a (black-box) monotonic classifier. These novel algorithms are polynomial in the run time complexity of the classifier and the number of features. Furthermore, the paper presents a practically efficient model-agnostic algorithm for enumerating formal explanations.

1 Introduction

Monotonicity is an often required constraint in practical applications of machine learning. Broadly, a monotonicity constraint requires that increasing (resp. decreasing) the value of one or more features, while keep the other features constant, will not cause the prediction to decrease (resp. increase). Monotonicity has been investigated in the context of classification (Cano et al., 2019), including neural networks (Sill, 1997; Magdon-Ismail & Sill, 2008; Bonakdarpour et al., 2018; Sivaraman et al., 2020; Liu et al., 2020), random forests (Bartley et al., 2019) and rule ensembles (Bartley et al., 2018), decision trees (Ben-David et al., 1989; Ben-David, 1995), decision lists (Potharst & Bioch, 2000) and decision rules (Verbeke et al., 2017), support vector machines (Bartley et al., 2016), nearest-neighbor classifiers (Duivesteijn & Feelders, 2008), among others (Fard et al., 2016; Gupta et al., 2016; You et al., 2017; Bonakdarpour et al., 2018). Monotonicity has been studied in bayesian networks (van der Gaag et al., 2004; Shih et al., 2018), active learning (Barile & Feelders, 2012) and, more recently, in fairness (Wang & Gupta, 2020).

To a much lesser extent, monotonicity has also been studied from the perspective of explainability, with one recent example being the study of the explainability of monotonic bayesian networks (Shih et al., 2018). This work proposes to compile different families of bayesian networks, including naive bayes and monotonic networks, into a decision diagram, which can then be used for computing PI-explanations[2]. Approaches based on an intermediate (knowledge) compilation step are characterized by two main drawbacks, namely their worst-case complexity, which is exponential both in time and in the size of the representation, but also the fact that these approaches are not model-agnostic, i.e. some formal logic representation of the model must be known and reasoned about. Clearly, model-agnostic heuristic approaches, which include LIME (Ribeiro et al., 2016), SHAP (Lundberg & Lee, 2017), or Anchor (Lundberg & Lee, 2017), can also be applied to explaining monotonic classifiers. However, these approaches do not readily exploit monotonicity, and both the theoretical and practical performance may be discouraging[3]. Furthermore, heuristic approaches offer no formal guarantees of rigor, e.g. an Anchor explanation may be consistent with points in feature space for which the model's prediction differ from the target prediction (Ignatiev, 2020).

On a more positive note, recent work proposed polynomial-time exact algorithms for computing PI-explanations explanations of different classes of classifiers (Marques-Silva et al., 2020), namely linear and naive bayes classifiers. These results were complemented by the observation that, for ML models related with some classes of knowledge representation languages, PI-explanations can also be computed in polynomial time (Audemard et al., 2020).

This paper extends these initial results to the case of monotonic classifiers, in a number of ways. First, the paper proposes model-agnostic algorithms for computing PI-explanations and contrastive explanations (Miller, 2019) for

[*]Equal contribution [1]IRIT, CNRS, Université Paul Sabatier, Toulouse, France [2]Monash University, Melbourne, Australia [3]VMware Research, CA, USA. Correspondence to: Joao Marques-Silva <joao.marques-silva@irit.fr>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[2]Given some feature space point $\mathbf{v}$, a PI-explanation is a subset-minimal subset of features which, the assignment of the corresponding coordinate value in $\mathbf{v}$, is sufficient for the prediction.

[3]In fact, there are recent negative results on the tractability of exact SHAP learning (Van den Broeck et al., 2020).

any monotonic ML model. Second, the complexity of the proposed algorithms is shown to be polynomial on the time required to run the (black-box) monotonic classifier and the number of features. Third, the paper proposes an algorithm for the iterative enumeration of formal explanations[4]. (This algorithm is worst-case exponential, but it is shown to be remarkably efficient in practice.)

The paper is organized as follows. Section 2 introduces the notation and definitions used in the rest of the paper. Section 3 details algorithms for computing one or more formal explanations of monotonic classifiers. Section 4 summarizes initial experiments, which confirm the scalability of the proposed algorithms. The paper concludes in Section 5.

2 Preliminaries

Classification problems. A classification problem is defined on a set of features (or attributes) $\mathcal{F} = \{1, \ldots, N\}$ and a set of classes $\mathcal{K} = \{c_1, c_2, \ldots, c_M\}$. Each feature $i \in \mathcal{F}$ takes values from a domain $\mathbb{D}_i$. Domains are bounded and ordered, and each domain can be defined on boolean, integer or real values. If $x_i \in \mathbb{D}_i$, then $\lambda(i)$ and $\mu(i)$ denote respectively the smallest and largest values that x_i can take, i.e. $\lambda(i) \leq x_i \leq \mu(i)$. Feature space is defined as $\mathbb{F} = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_N$. The notation $\mathbf{x} = (x_1, \ldots, x_N)$ denotes an arbitrary point in feature space, where each x_i is a variable taking values from $\mathbb{D}_i$. Moreover, the notation $\mathbf{v} = (v_1, \ldots, v_N)$ represents a specific point in feature space, where each v_i is a constant representing one concrete value from $\mathbb{D}_i$. An *instance* (or example) denotes a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$. (We also use the term *instance* to refer to $\mathbf{v}$, leaving c implicit.) An ML classifier $\mathbb{C}$ is characterized by a *classification function* κ that maps feature space $\mathbb{F}$ into the set of classes $\mathcal{K}$, i.e. $\kappa : \mathbb{F} \to \mathcal{K}$.

Monotonic classification. Given two points in feature space $\mathbf{a}$ and $\mathbf{b}$, $\mathbf{a} \leq \mathbf{b}$ if $a_i \leq b_i$, for all $i \in \{1, \ldots, N\}$. A set of classes $\mathcal{K} = \{c_1, \ldots, c_M\}$ is *ordered* if it respects a total order $\preccurlyeq$, with $c_1 \preccurlyeq c_2 \preccurlyeq \ldots \preccurlyeq c_M$. An ML classifier $\mathbb{C}$ is fully monotonic if the associated classification function is monotonic, i.e. $\mathbf{a} \leq \mathbf{b} \Rightarrow \kappa(\mathbf{a}) \preccurlyeq \kappa(\mathbf{b})$[5]. Throughout the paper, when referring to a monotonic classifier, this signifies a fully monotonic classifier. In addition, the interaction with a classifier is restricted to computing the value of $\kappa(\mathbf{v})$, for some point $\mathbf{v} \in \mathbb{F}$, i.e. the classifier will be viewed as a black-box.

Example 1 (Running example)**.** *Let us consider a classifier for predicting student grades. We assume that the classifier has learned the following formula (after being trained with grades of students from different cohorts):*

$$\begin{aligned} S &= \max[0.3 \times Q + 0.6 \times X + 0.1 \times H, R] \\ M &= \text{ite}(S \geq 9, A, \text{ite}(S \geq 7, B, \text{ite}(S \geq 5, C, \\ &\quad \text{ite}(S \geq 4, D, \text{ite}(S \geq 2, E, F))))) \end{aligned}$$

S, Q, X, H and R denote, respectively, the final score, the marks on the quiz, the exam, the homework, and the mark of an optional research project. Each mark ranges from 0 to 10. (For the optional mark R, the final mark is 0 if the student opts out.) The final score is the largest of the two marks, as shown above. Moreover, the final grade M is defined using an ite *(if-then-else) operator, and ranges from A to F. As a result, Q, X, H and R represent the features of the classification problem, respectively numbered 1, 2, 3 and 4, and so $\mathcal{F} = \{1, 2, 3, 4\}$. Each feature takes values from $[0, 10]$, i.e. $\lambda(i) = 0$ and $\mu(i) = 10$. The set of classes is $\mathcal{K} = \{A, B, C, D, E, F\}$, with $F \preccurlyeq E \preccurlyeq D \preccurlyeq C \preccurlyeq B \preccurlyeq A$. Clearly, the complete classifier (that given the different marks computes a final grade) is monotonic. Moreover, we will we consider a specific point of feature space representing student s_1, $(Q, X, H, R) = (10, 10, 5, 0)$, with a predicted grade of A, i.e. $\kappa(10, 10, 5, 0) = A$.*

Abductive and contrastive explanations. We now define formal explanations. Prime implicant (PI) explanations (Shih et al., 2018) denote a minimal set of literals (relating a feature value x_i and a constant v_i from its domain $\mathbb{D}_i$) that are sufficient for the prediction[6]. Formally, given $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a PI-explanation (AXp) is any minimal subset $\mathcal{X} \subseteq \mathcal{F}$ such that,

$$\forall(\mathbf{x} \in \mathbb{F}).\ \left[\bigwedge\nolimits_{i \in \mathcal{X}}(x_i = v_i)\right] \rightarrow (\kappa(\mathbf{x}) = c) \qquad (1)$$

AXp's can be viewed as answering a 'Why?' question, i.e. why is some prediction made given some point in feature space. A different view of explanations is a contrastive explanation (Miller, 2019), which answers a 'Why Not?' question, i.e. which features can be changed to change the prediction. A formal definition of contrastive explanation is proposed in recent work (Ignatiev et al., 2020). Given $\mathbf{v} = (v_1, \ldots, v_N) \in \mathbb{F}$ with $\kappa(\mathbf{v}) = c$, a CXp is any minimal subset $\mathcal{Y} \subseteq \mathcal{F}$ such that,

$$\exists(\mathbf{x} \in \mathbb{F}).\ \bigwedge\nolimits_{j \in \mathcal{F} \setminus \mathcal{Y}}(x_j = v_j) \land (\kappa(\mathbf{x}) \neq c) \qquad (2)$$

Building on the results of R. Reiter in model-based diagnosis (Reiter, 1987), (Ignatiev et al., 2020) proves a minimal hitting set (MHS) duality relation between AXp's and CXp's, i.e. AXp's are MHSes of CXp's and vice-versa.

[4]The term formal explanation is used in contrast with heuristic explanation (Ribeiro et al., 2016; Lundberg & Lee, 2017; Ribeiro et al., 2018) and it will be defined precisely in Section 2.

[5]The paper adopts the classification of monotonic classifiers proposed in earlier work (Daniels & Velikova, 2010).

[6]PI-explanations are related with abduction, and so are also referred to as abductive explanations (AXp) (Ignatiev et al., 2019). More recently, PI-explanations have been studied from a knowledge compilation perspective (Audemard et al., 2020).

Example 2 (AXp's & CXp's)**.** *As can be readily observed (from the expression for M in Example 1), as long as Q and X take value 10, the prediction will be A,* independently *of the values given to H and R. Hence, given $(Q, X, H, R) = (10, 10, 5, 0)$, one AXp is $\{1, 2\}$. Moreover, to obtain a different prediction, it suffices to allow a suitable change of value in Q (or alternatively in X). Hence, given $(Q, X, H, R) = (10, 10, 5, 0)$, one CXp is $\{1\}$ (and another is $\{2\}$). As can be observed, $\{1, 2\}$ is the only MHS of $\{\{1\}, \{2\}\}$ and vice-versa. These are the only AXp's and CXp's for the example instance.*

Despite being characterized by a formal guarantee of rigor, abductive and contrastive explanations also exhibit a number of drawbacks[7]. First, scalability can be an issue, and that explains recent efforts on identifying classes of classifiers for which explanations can be computed in polynomial time (Marques-Silva et al., 2020; Izza et al., 2020; Shi et al., 2020; Audemard et al., 2020; 2021; Huang et al., 2021), or classes of classifiers that can be explained efficiently in practice (Ignatiev, 2020; Choi et al., 2020; Izza & Marques-Silva, 2021; Ignatiev & Marques-Silva, 2021). Second, in some settings, the guarantee of rigor that characterizes model-accurate approaches, may in fact be unnecessary. Until recently, explanations exhibiting probabilistic guarantees of rigor were largely non-existing. However, there is recent work on computing explanations with probabilistic guarantees (Wäldchen et al., 2021; Izza et al., 2021). Third, whereas heuristic explanation approaches are distribution-aware (Ribeiro et al., 2016; Lundberg & Lee, 2017; Ribeiro et al., 2018), model-accurate explanation approaches are not. Nevertheless, recent work proposed to exploit input constraints as a mechanism to address input distributions (Gorji & Rubin, 2021). Fourth, in some settings users may prefer explanations that relate groups of features. This paper addresses the first drawback, and proposes efficient algorithms for explaining monotonic classifiers.

Boolean satisfiability (SAT). SAT is the decision problem for propositional logic. The paper uses standard notation and definitions e.g. (Biere et al., 2009). A propositional formula is defined on a set U of boolean variables, where the domain of each variable $u_i \in U$ is $\{0, 1\}$. We consider conjunctive normal form (CNF) formulas, where a formula is a conjunction of clauses, each clause is a disjunction of literals, and a literal is a variable u_i or its negation $\neg u_i$. CNF formulas and SAT reasoners are used in Section 3.2.

3 Explanations for Monotonic Classifiers

This section describes three algorithms. The first algorithm serves to compute one AXp (and is referred to as findAXp).

[7] In some settings, these drawbacks justify why model-agnostic explanations may be a viable alternative.

Algorithm 1 Finding one AXp – findAXp($\mathcal{F}, \mathcal{S}, \mathbf{v}$)

$\mathbf{v}_\mathrm{L} \leftarrow (v_1, \ldots, v_N)$
$\mathbf{v}_\mathrm{U} \leftarrow (v_1, \ldots, v_N)$ // Ensures: $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$
$(\mathcal{C}, \mathcal{D}, \mathcal{P}) \leftarrow (\mathcal{F}, \emptyset, \emptyset)$
for all $i \in \mathcal{S}$ **do**
  $(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D}) \leftarrow \mathsf{FreeAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D})$
end for // Require: $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$, given $\mathcal{S}$
for all $i \in \mathcal{F} \setminus \mathcal{S}$ **do** // Loop inv.: $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$
  $(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D}) \leftarrow \mathsf{FreeAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D})$
  if $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$ **then** // If invariant broken, fix it
    $(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{D}, \mathcal{P}) \leftarrow \mathsf{FixAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{D}, \mathcal{P})$
  end if
end for
return $\mathcal{P}$

Its complexity is polynomial in the run time complexity of the classifier. The second algorithm serves to compute one CXp (and is referred to as findCXp). It has the same polynomial complexity as findAXp. The third algorithm shows how to use SAT reasoners for iteratively enumerating AXp's or CXp's. This algorithm is inspired by earlier work (Liffiton et al., 2016), but with key observations that minimize the number of times a SAT reasoner is called. This algorithm is based on the other two algorithms, and is described in Section 3.2.

One key property of the three algorithms is that, besides knowing that the classifier is monotonic, *no* additional information about the classifier is required. Indeed, the algorithms described in this section only require running the classifier for specific points in feature space. Thus, and similarly to LIME (Ribeiro et al., 2016), SHAP (Lundberg & Lee, 2017) or Anchor (Ribeiro et al., 2018), the algorithms proposed in this section are model-agnostic. However, and in contrast also with LIME, SHAP or Anchor, the proposed algorithms compute rigorously defined AXp's, CXp's, and also serve for the enumeration of explanations.

3.1 Finding One AXp and One CXp

The two algorithms findAXp and findCXp (shown as Algorithm 1 and Algorithm 2) share a number of common concepts, while solving different problems. These concepts are summarized next. The two algorithms iteratively update three sets of features ($\mathcal{C}$, $\mathcal{D}$ and $\mathcal{P}$) and two points in feature space ($\mathbf{v}_\mathrm{L}$ and $\mathbf{v}_\mathrm{U}$). Using these variables, the two algorithms maintain two invariants. The first invariant is that $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{P}$ form a partition of $\mathcal{F}$, and represent respectively the candidate, dropped and picked sets of features (with the picked features denoting those that are included either in an AXp or an CXp). The second invariant serves to ensure that the selected set of features satisfies (1) (for findAXp) or (2) (for findCXp). Maintaining this invariant, requires

Algorithm 2 Finding one CXp – findCXp($\mathcal{F}, \mathcal{S}, \mathbf{v}$)

1: $\mathbf{v}_\mathrm{L} \leftarrow (\lambda(1), \ldots, \lambda(N))$
2: $\mathbf{v}_\mathrm{U} \leftarrow (\mu(1), \ldots, \mu(N))$ // Ensures: $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$
3: $(\mathcal{C}, \mathcal{D}, \mathcal{P}) \leftarrow (\mathcal{F}, \emptyset, \emptyset)$
4: **for all** $i \in \mathcal{S}$ **do**
5: $\quad(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D}) \leftarrow \mathsf{FixAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D})$
6: **end for** // Require: $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$, given $\mathcal{S}$
7: **for all** $i \in \mathcal{F} \setminus \mathcal{S}$ **do** // Loop inv.: $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$
8: $\quad(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D}) \leftarrow \mathsf{FixAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{C}, \mathcal{D})$
9: $\quad$**if** $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$ **then** // If invariant broken, fix it
10: $\quad\quad(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{D}, \mathcal{P}) \leftarrow \mathsf{FreeAttr}(i, \mathbf{v}, \mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{D}, \mathcal{P})$
11: $\quad$**end if**
12: **end for**
13: **return** $\mathcal{P}$

iteratively updating two points $\mathbf{v}_\mathrm{L} = (v_{\mathrm{L}_1}, \ldots, v_{\mathrm{L}_N})$ and $\mathbf{v}_\mathrm{U} = (v_{\mathrm{U}_1}, \ldots, v_{\mathrm{U}_N})$, denoting respectively lower and upper bounds on the class values that can be obtained given the features that are allowed to take any value in their domain.

Finding one AXp. We detail below the main steps of algorithm findAXp (see Algorithm 1). (Lines 4 to 5 are used for enumerating explanations, and so we assume $\mathcal{S} = \emptyset$ for now.) The main goal of findAXp is to find a *maximal* set of features $\mathcal{D}$ which are allowed to take *any* value, i.e. that are *free*. For such a set $\mathcal{D}$, the set of features that remain fixed to the value specified in $\mathbf{v}$, i.e. $\mathcal{P} = \mathcal{F} \setminus \mathcal{D}$, is a minimal set of (picked) features that is sufficient for the prediction, as intended. The different sets used by the algorithm are initialized in line 3. (As noted earlier, the sets $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{P}$ form a partition of $\mathcal{F}$, and $\mathcal{C} = \emptyset$ upon termination.)

For findAXp, the second invariant of the algorithm is that $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$, i.e. by allowing the features in $\mathcal{P} \cup \mathcal{C}$ to take the corresponding value in $\mathbf{v}$, the value of the prediction is guaranteed not to change.

The use of the second invariant $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$ is justified by the following result.

Proposition 1. *If $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$, then it holds that, $\forall(\mathbf{x} \in \mathbb{F}).[\mathbf{v}_\mathrm{L} \leq \mathbf{x} \leq \mathbf{v}_\mathrm{U}] \rightarrow [\kappa(\mathbf{x}) = \kappa(\mathbf{v})]$.*

The algorithm starts by enforcing the second invariant as the result of executing lines 1 and 2.

Moreover, findAXp analyzes one feature at a time. Starting from the set $\mathcal{C}$ of candidate features (in line 7), the algorithm iteratively picks a feature i from $\mathcal{C}$ and makes a decision about whether to drop the feature from the explanation. The first step is to assume that the feature i can indeed be allowed to take any value. This is done in line 8, by calling the following function FreeAttr:

$\mathbf{v}_\mathrm{L} \leftarrow (v_{\mathrm{L}_1}, \ldots, \lambda(i), \ldots, v_{\mathrm{L}_N})$
$\mathbf{v}_\mathrm{U} \leftarrow (v_{\mathrm{U}_1}, \ldots, \mu(i), \ldots, v_{\mathrm{U}_N})$
$(\mathcal{A}, \mathcal{B}) \leftarrow (\mathcal{A} \setminus \{i\}, \mathcal{B} \cup \{i\})$
return $(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{A}, \mathcal{B})$

where $\mathcal{A}$ is replaced by $\mathcal{C}$ and B is replaced by $\mathcal{D}$, and so feature i is moved from $\mathcal{C}$ to $\mathcal{D}$. In addition, the value of i is now allowed to range from $\lambda(i)$ (in $\mathbf{v}_\mathrm{L}$) to $\mu(i)$ (in $\mathbf{v}_\mathrm{U}$),

The next step of the algorithm (in line 9) is to decide whether allowing i to take any value breaks the invariant $\kappa(\mathbf{v}_\mathrm{L}) = \kappa(\mathbf{v}_\mathrm{U})$. If the invariant is not broken, then the algorithm moves to analyze the next feature (in line 7). However, if the invariant is broken, then the the feature cannot take any value, and so it must be fixed to the corresponding value in $\mathbf{v}$. This is done by calling (in line 10) the following function FixAttr:

$\mathbf{v}_\mathrm{L} \leftarrow (v_{\mathrm{L}_1}, \ldots, v_i, \ldots, v_{\mathrm{L}_N})$
$\mathbf{v}_\mathrm{U} \leftarrow (v_{\mathrm{U}_1}, \ldots, v_i, \ldots, v_{\mathrm{U}_N})$
$(\mathcal{A}, \mathcal{B}) \leftarrow (\mathcal{A} \setminus \{i\}, \mathcal{B} \cup \{i\})$
return $(\mathbf{v}_\mathrm{L}, \mathbf{v}_\mathrm{U}, \mathcal{A}, \mathcal{B})$

where $\mathcal{A}$ is replaced by $\mathcal{D}$ and $\mathcal{B}$ is replaced by $\mathcal{P}$, and so feature i is moved from $\mathcal{D}$ to $\mathcal{P}$. In addition, the value of i is once again fixed to the corresponding value in $\mathbf{v}$. After analyzing all features, the algorithm findAXp terminates (in line 13) by return the (minimal) set of features $\mathcal{P}$ that are fixed to their value in $\mathbf{v}$. It is immediate to conclude that each feature is analyzed once, and that for each feature, the classifier is invoked twice. Given the discussion above, we conclude that,

Theorem 1. *Given a monotonic classifier, an instance $\mathbf{v}$ with prediction $c = \kappa(\mathbf{v})$, Algorithm 1 computes one AXp in linear time in the running time complexity of the classifier.*

We illustrate the operation of findAXp, with an example.

Example 3. *Given the monotonic classifier from Example 1, and the concrete case of student s_1, with $(Q, X, H, R) = (10, 10, 5, 0)$ and predicted mark A, we show how one PI-explanation can computed. (In settings with more than one AXp, changing the order of how features are analyzed, may results in a different explanation being obtained.) For each feature i, $1 \leq i \leq 4$, $\lambda(i) = 0$ and $\mu(i) = 10$. Moreover, features are analyzed in order: $\langle 1, 2, 3, 4 \rangle$; the order is arbitrary. The algorithm's execution is summarized in Table 1. As can be observed, features 1 and 2 are kept as part of the PI-explanation (decision is ✓ in line 9, i.e. invariant is broken and features are kept), whereas features 3 and 4 are dropped from the PI-explanation (decision is ✗, i.e. invariant holds). As a result, the PI-explanation for the grade of student s_1 is $\{1, 2\}$, which denotes that as long as $(Q = 10) \land (X = 10)$, the prediction will be A.*

Finding one CXp. The two algorithms findAXp and findCXp are organized in a similar way. (This in part results from the fact that AXps are minimal hitting sets of CXps and vice-versa (Ignatiev et al., 2020).) We briefly explain

Feat.	Initial values		Changed values		Predictions		Dec.	Resulting values	
	$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$	$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$	$\kappa(\mathbf{v}_\mathrm{L})$	$\kappa(\mathbf{v}_\mathrm{U})$		$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$
1	(10,10,5,0)	(10,10,5,0)	(0,10,5,0)	(10,10,5,0)	C	A	✓	(10,10,5,0)	(10,10,5,0)
2	(10,10,5,0)	(10,10,5,0)	(10,0,5,0)	(10,10,5,0)	E	A	✓	(10,10,5,0)	(10,10,5,0)
3	(10,10,5,0)	(10,10,5,0)	(10,10,0,0)	(10,10,5,0)	A	A	✗	(10,10,0,0)	(10,10,10,0)
4	(10,10,0,0)	(10,10,10,0)	(10,10,0,0)	(10,10,10,10)	A	A	✗	(10,10,0,0)	(10,10,10,10)

Table 1: Execution of algorithm for finding one AXp

Feat.	Initial values		Changed values		Predictions		Dec.	Resulting values	
	$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$	$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$	$\kappa(\mathbf{v}_\mathrm{L})$	$\kappa(\mathbf{v}_\mathrm{U})$		$\mathbf{v}_\mathrm{L}$	$\mathbf{v}_\mathrm{U}$
1	(0,0,0,0)	(10,10,10,10)	(10,0,0,0)	(10,10,10,10)	E	A	✗	(10,0,0,0)	(10,10,10,10)
2	(10,0,0,0)	(10,10,10,10)	(10,10,0,0)	(10,10,10,10)	A	A	✓	(10,0,10,0)	(10,10,10,10)
3	(10,0,0,0)	(10,10,10,10)	(10,0,5,0)	(10,10,5,10)	E	A	✗	(10,0,5,0)	(10,0,5,10)
4	(10,0,5,0)	(10,10,5,10)	(10,0,5,0)	(10,10,5,0)	E	A	✗	(10,0,5,0)	(10,10,5,0)

Table 2: Execution of algorithm for finding one CXp

the differences when computing a CXp (see Algorithm 2). (Lines 4 to 5 are used for enumerating explanations, and so we assume $\mathcal{S} = \emptyset$ for now.)

The main goal of findCXp is to find a *maximal* set of features $\mathcal{D}$ that are only allowed to take the value specified in $\mathbf{v}$, i.e. that are *fixed*. For such a set $\mathcal{D}$, the set of features that are allowed to take any value, i.e. $\mathcal{P} = \mathcal{F} \setminus \mathcal{D}$, is a minimal set that, by being allowed to take any value in their domain, suffices for allowing the prediction to change, as intended. The different sets used by the algorithm are initialized in line 3.

For findCXp, the second invariant of the algorithm is that $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$, i.e. by allowing the features in $\mathcal{P} \cup \mathcal{C}$ to take any value, the value of the prediction does not change. The algorithm starts by enforcing the second invariant as the result of executing lines 1 and 2.

The use of the second invariant $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$ is justified by the following result.

Proposition 2. *If $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$, then it holds that, $\exists(\mathbf{x} \in \mathbb{F}).[\mathbf{v}_\mathrm{L} \leq \mathbf{x} \leq \mathbf{v}_\mathrm{U}] \land [\kappa(\mathbf{x}) \neq \kappa(\mathbf{v})]$.*

Similarly to findAXp, findCXp analyzes one feature at a time. Starting from the set $\mathcal{C}$ of candidate features (in line 7), the algorithm iteratively picks a feature i from $\mathcal{C}$ and makes a decision about whether to drop the feature from the explanation. The first step is to assume that the feature i can indeed be fixed to the corresponding value in $\mathbf{v}$. This is done in line 8, by calling the following function FixAttr, where $\mathcal{A}$ is replaced by $\mathcal{C}$, and B is replaced by $\mathcal{D}$, and so feature i is moved from $\mathcal{C}$ to $\mathcal{D}$. In addition, the value of i is now fixed to its value in $\mathbf{v}$.

The next step of the algorithm (in line 9) is to decide whether fixing the value of i breaks the invariant $\kappa(\mathbf{v}_\mathrm{L}) \neq \kappa(\mathbf{v}_\mathrm{U})$. If the invariant is not broken, then the algorithm moves to analyze the next feature (in line 7). However, if the invariant is broken, then the feature cannot be fixed, and so it must be allowed to take any value from its domain. This is done by calling (in line 10) the following function FreeAttr, with $\mathcal{A}$ replaced by $\mathcal{D}$ and $\mathcal{B}$ replaced by $\mathcal{P}$, and so feature i is moved from $\mathcal{D}$ to $\mathcal{P}$. In addition, the value of i is once again allowed to take any value from its domain. After analyzing all features, the algorithm findCXp terminates (in line 13) by returning the (minimal) set of features $\mathcal{P}$ that are allowed to take any value from their domain. It is immediate to conclude that each feature is analyzed once, and that for each feature, the classifier is invoked twice. Given the discussion above, we conclude that,

Theorem 2. *Given a monotonic classifier, an instance $\mathbf{v}$ with prediction $c = \kappa(\mathbf{v})$, Algorithm 2 computes one CXp in linear time in the running time complexity of the classifier.*

We illustrate the operation of findCXp, with an example.

Example 4. *For the running example (see Examples 1, 2 and 3), for instance $\mathbf{v}_0 = (10, 10, 5, 0)$ with prediction A, we illustrate the computation of one CXp. The algorithm's execution is summarized in Table 2. (When computing one CXp, a feature is* kept *(decision is ✓) if it is declared free, and it is* dropped *(decision is ✗) if it must be fixed.) As can be observed, a contrastive explanation is: $\{2\}$, i.e. there is*

an assignment to feature 2 (i.e. to X), which guarantees a change of prediction when the other features are kept to their values. For example, by setting $X = 0$ (and keeping the remaining values fixed), the value of the prediction changes.

Complexity. As can be readily concluded from Algorithm 1 and Algorithm 2, the algorithms execute in linear time in the number of features. However, in each iteration of the algorithm, the classifier is invoked twice, for finding the predicted classes for $\mathbf{v}_{\mathrm{L}}$ and for $\mathbf{v}_{\mathrm{U}}$. We will represent the time required by the classifier as $\mathcal{T}_{\mathbb{C}}$, and so the overall run time of each algorithm is $\mathcal{O}(|\mathcal{F}| \times \mathcal{T}_{\mathbb{C}})$.

3.2 Enumerating Explanations

We first show that for monotonic classifiers, the enumeration of explanations with polynomial-time delay is computationally hard.

Theorem 3. *Determining the existence of $\lfloor N/2 \rfloor + 1$ AXp's (or CXp's) of a monotonic N-feature classifier is NP-complete.*

(The proof is included in the supplementary material.) Since the enumeration of AXp's and CXp's with polynomial delay is unlikely, we describe in this section how to use SAT reasoners for the enumeration of AXp's and CXp's of a monotonic classifier. (Although we prove the algorithm to be sound and complete, the algorithm necessarily has leeway in selecting the order in which AXp's and CXp's are listed.) The algorithm uses the following propositional representation:

1. The algorithm will iteratively add clauses to a CNF formula $\mathcal{H}$. The clauses in $\mathcal{H}$ account for the AXp's and CXp's already computed, and serve to prevent their repetition.
2. Formula $\mathcal{H}$ is defined on a set of variables u_i, $1 \leq i \leq n$, where each u_i denotes whether feature i is declared free ($u_i = 1$) or is alternatively declared fixed ($u_i = 0$).

The algorithm proposed in this section requires exactly one call to a SAT reasoner before computing one explanation (either AXp/CXp), and one additional call to decide that all explanations have been computed. As a result, the number of calls to a SAT reasoner is $|\text{AXp}|+|\text{CXp}|+1$. Furthermore, the size of the formula grows by one clause after each AXp or CXp is computed. In practice, for a wide range of ML settings, both the number of variables and the number of clauses are well within the reach of modern SAT reasoners.

Proposition 3. *Let $\mathbf{v}$ be a point in feature space, let $\kappa(\mathbf{v}) = c \in \mathcal{K}$, and let $\mathcal{Z} \subseteq \mathcal{F}$. Then, either (1) (on page 2) holds, with $\mathcal{X} = \mathcal{Z}$, or (2) (also on page 2) holds, with $\mathcal{Y} = \mathcal{F} \setminus \mathcal{Z}$, but not both.*

Proposition 3 essentially states that, given a set $\mathcal{Z}$ of features, if these are fixed, and the others are allowed to take any value from their domains, then either the prediction never changes, or there exists an assignment to the non-fixed features, which causes the prediction to change. The approach for enumerating AXp's and CXp's is shown in Algorithm 3. The algorithm starts in line 1 by initializing the CNF formula $\mathcal{H}$ without clauses (these will be added as the algorithm executes). The main loop (from line 2 to line 19) is executed while the formula $\mathcal{H}$ is satisfiable. This is decided with a call to a SAT reasoner (in line 3). Any satisfying assignment to the formula $\mathcal{H}$ partitions the features into two sets: one denoting the features that can take any value (with $u_i = 1$) and another denoting the features that take the corresponding value in $\mathbf{v}$ (with $u_i = 0$). (The assignment effectively identifies a set $\mathcal{Z} \subseteq \mathcal{F}$, of fixed features, and thus we can invoke Proposition 3.) In line 5 and line 6, the algorithm creates $\mathbf{v}_{\mathrm{L}}$ and $\mathbf{v}_{\mathrm{U}}$. For a fixed feature i, both $\mathbf{v}_{\mathrm{L}}$ and $\mathbf{v}_{\mathrm{U}}$ are assigned value v_i. For a free feature i, $\mathbf{v}_{\mathrm{L}}$ and $\mathbf{v}_{\mathrm{U}}$ are respectively assigned to $\lambda(i)$ and $\mu(i)$. Let $\mathcal{Z}$ denote the set of fixed features. In line 7, we check in which case of Proposition 3 applies.

If $\kappa(\mathbf{v}_{\mathrm{L}}) = \kappa(\mathbf{v}_{\mathrm{U}})$, then we know that the invariant of Algorithm 1 holds. Moreover, $\mathcal{F} \setminus \mathcal{Z}$ is a subset of an AXp. Hence, we set $\mathcal{S} = \mathcal{F} \setminus \mathcal{Z}$ as the *seed* for $\mathsf{findAXp}$. This is shown in lines 8 and 9. After reporting the computed AXp, represented by the set of features $\mathcal{P}$, we prevent the same AXp from being computed again by requiring that at least one of the fixed features must be free in future satisfying assignments of $\mathcal{H}$. This is represented as a positive clause.

Proposition 4. *In the case $\kappa(\mathbf{v}_{\mathrm{L}}) = \kappa(\mathbf{v}_{\mathrm{U}})$, set $\mathcal{S}$ is such that, for any previously computed AXp, at least one feature*

Algorithm 3 Enumeration of AXp's and CXp's

1: $\mathcal{H} \leftarrow \emptyset$ // $\mathcal{H}$ defined on set U
2: **repeat**
3: $(\mathsf{outc}, \mathbf{u}) \leftarrow \mathsf{SAT}(\mathcal{H})$
4: **if** $\mathsf{outc} =$ **true then**
5: $\mathbf{v}_{\mathrm{L}} \leftarrow (v_{\mathrm{L}_1}, \ldots, v_{\mathrm{L}_N})$, s.t. $v_{\mathrm{L}_i} \leftarrow \text{ite}(u_i, \lambda(i), v_i)$
6: $\mathbf{v}_{\mathrm{U}} \leftarrow (v_{\mathrm{U}_1}, \ldots, v_{\mathrm{U}_N})$, s.t. $v_{\mathrm{U}_i} \leftarrow \text{ite}(u_i, \mu(i), v_i)$
7: **if** $\kappa(\mathbf{v}_{\mathrm{L}}) = \kappa(\mathbf{v}_{\mathrm{U}})$ **then**
8: $\mathcal{S} \leftarrow \{i \in \mathcal{F} \,|\, u_i = 1\}$ // $\mathcal{F} \setminus \mathcal{S} \supseteq$ some AXp
9: $\mathcal{P} \leftarrow \mathsf{findAXp}(\mathcal{F}, \mathcal{S}, \mathbf{u})$
10: $\mathsf{reportAXp}(\mathcal{P})$
11: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in \mathcal{P}} u_i)\}$
12: **else**
13: $\mathcal{S} \leftarrow \{i \in \mathcal{F} \,|\, u_i = 0\}$ // $\mathcal{F} \setminus \mathcal{S} \supseteq$ some CXp
14: $\mathcal{P} \leftarrow \mathsf{findCXp}(\mathcal{F}, \mathcal{S}, \mathbf{u})$
15: $\mathsf{reportCXp}(\mathcal{P})$
16: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\vee_{i \in \mathcal{P}} \neg u_i)\}$
17: **end if**
18: **end if**
19: **until** $\mathsf{outc} =$ **false**

will be included in $\mathcal{S}$ (as a free literal). Since findAXp *only grows $\mathcal{S}$, then the Algorithm 3 does not repeat AXp's.*

Moreover, if $\kappa(\mathbf{v}_{\mathrm{L}}) \neq \kappa(\mathbf{v}_{\mathrm{U}})$, then we know that the invariant of Algorithm 2 holds. Moreover, $\mathcal{Z}$ is a subset of a CXp. Hence, we set $\mathcal{S} = \mathcal{Z}$ as the *seed* for findCXp. This is shown in lines 13 and 14. After reporting the computed CXp, represented by the set of features $\mathcal{P}$, we prevent the same CXp from being computed by requiring that at least one of the free features must be free in future satisfying assignments of $\mathcal{H}$. This is represented as negative clause.

Proposition 5. *In the case $\kappa(\mathbf{v}_{\mathrm{L}}) \neq \kappa(\mathbf{v}_{\mathrm{U}})$, set $\mathcal{S}$ is such that, for any previously computed CXp, at least one feature will be included in $\mathcal{S}$ (as a fixed literal). Since* findCXp *only grows $\mathcal{S}$, then the Algorithm 3 does not repeat CXp's.*

Given the above, and since the number of AXp's and CXp's (being subsets of $\mathcal{F}$) is finite, then we have,

Theorem 4. *Algorithm 3 is sound and complete for the enumeration of AXp's and CXp's.*

Example 5. *Building on earlier examples, we summarize the main steps of the SAT oracle-based algorithm for enumerating AXp and CXp explanations. Table 3 illustrates one execution of the proposed algorithm. There are 1 AXp's and 2 CXp's. (Regarding the call to the SAT oracle, the satisfying assignments shown are intended to be as arbitrary as possible, given the existing constraints; other satisfying assignments could have been picked.) For each computed AXp, we add to $\mathcal{H}$ one positive clause. In this example, we add the clause $(u_1 \vee u_2)$, since the AXp is $\{1, 2\}$. By adding this clause, we guarantee that features 1 and 2 will not both be deemed fixed by subsequent satisfying assignments of $\mathcal{H}$. Similarly, for each computed CXp, we add to $\mathcal{H}$ one negative clause. For the example, the clauses added are $(\neg u_1)$ for CXp $\{1\}$, and $(\neg u_2)$ for CXp $\{2\}$. In both cases, the added clause guarantees that feature 1 (resp. 2) will not be deemed free by subsequent satisfying assignments of $\mathcal{H}$. One additional observation is that the number of SAT oracle calls matches the number of AXp's plus the number of CXp's and plus one final call to terminate the algorithm's execution. For step 4 of the algorithm, it is easy to conclude that $\mathcal{H}$ is unsatisfiable, as intended.*

3.3 Related Work

The algorithms for computing one AXp or one CXp for a monotonic classifier are novel. However, the insight of analyzing elements of a set (i.e. features in our case) to find a minimal set respecting some property has been studied in a vast number of settings (e.g. (Chinneck, 2008) for an overview). The proposed solution for reasoning about features that can take boolean, integer or real values, represents another aspect of novelty. In the case of monotonic classifiers, we obtain a running time that is linear in the running time complexity of the classifier. This result applies in the case of *any* monotonic classifier, and so it improves significantly over the worst-case exponential time and space approach proposed in earlier work (Shih et al., 2018), for the concrete case of monotonic bayesian networks. The algorithm for enumerating AXp's and CXp's for a monotonic classifier is also novel. However, it is inspired by the MARCO algorithm for the analysis of inconsistent logic theories (Liffiton et al., 2016). Although MARCO can be optimized in different ways, Algorithm 3 can be related with its most basic formulation. Since computing one AXp or one CXp can be achieved in polynomial time (conditioned by the classifier run time complexity), then our approach guarantees that exactly one SAT reasoner call is required for each computed minimal set (i.e. AXp or CXp in our case).

4 Experiments

The objective of this section is to illustrate the scalability of both the algorithms for finding one explanation, but also the algorithm for enumerating explanations. The tool `XMono` implements the algorithms 1, 2 and 3[8]. As observed in recent work, most monotonic classifiers are not publicly available (Cano et al., 2019)[9]. We analyze two publicly available classifiers, and describe two experiments. The first experiment evaluates `XMono` for explaining two recently proposed tools, COMET (Sivaraman et al., 2020) and monoboost[10] (Bartley et al., 2018). COMET is run on the Auto-MPG[11] dataset studied in earlier work (Sivaraman et al., 2020), with the choice justified by the time the classifier takes to run. monoboost is run on a monotonic dataset with two classes (as required by the tool) (Bartley et al., 2018). We use a monotonic subset (PimaMono) of the Pima dataset[12]. A second experiment compares `XMono` with Anchor (Ribeiro et al., 2018), both in terms of the number of calls to the classifier and running time, but also in terms of the quality of the computed explanations[13], namely accuracy and size. This second experiment also considers two datasets. The first dataset is BankruptcyRisk (Greco et al., 1998) (which is monotonic if one instance is dropped). For this dataset, the monotonic decision tree classifier proposed in earlier work is used (Potharst & Bioch, 2000). The second dataset is PimaMono, and the classifier used is the one obtained with monoboost (as in the first experiment). All

[8] `XMono` is available from `https://git.io/JZZBX`.

[9] One exception is TensorFlow (Abadi et al., 2016). Its integration with `XMono` is the subject of future work.

[10] Available from `https://git.io/JZZBx`.

[11] `http://tiny.cc/k3qytz`.

[12] `http://tiny.cc/l3qytz`.

[13] It should be underlined that neither Anchor (Ribeiro et al., 2018), LIME (Ribeiro et al., 2016) nor SHAP (Lundberg & Lee, 2017) can enumerate explanations, neither can these tools compute heuristic contrastive explanations.

Step	$\mathcal{H}$	$\mathbf{u}$ / out	$\mathbf{v}_{\mathrm{L}}$	$\mathbf{v}_{\mathrm{U}}$	$\kappa(\mathbf{v}_{\mathrm{L}})$	$\kappa(\mathbf{v}_{\mathrm{U}})$	AXp	CXp	Clause added
1	$\emptyset$	$(0,0,1,0)$	$(10,10,0,0)$	$(10,10,10,0)$	A	A	$\{1,2\}$	–	$(u_1 \lor u_2)$
2	$(u_1 \lor u_2)$	$(1,0,0,1)$	$(0,10,5,0)$	$(10,10,5,10)$	C	A	–	$\{1\}$	$(\neg u_1)$
3	$(u_1 \lor u_2)$ $(\neg u_1)$	$(0,1,1,0)$	$(10,0,0,0)$	$(10,10,10,0)$	E	A	–	$\{2\}$	$(\neg u_2)$
4	$(u_1 \lor u_2)$ $(\neg u_1), (\neg u_2)$	UNSAT	–	–	–	–	–	–	–

Table 3: Execution of enumeration algorithm

experiments were run on a MacBook Pro, with a 2.4GHz quad-core i5 processor, and 16 GByte of RAM, running MacOS Big Sur. For each dataset, we either pick 100 instances, randomly selected, or the total number of instances in the dataset, in case this number does not exceed 100.

4.1 Cost of Computing Explanations

We run XMono on a neural network classifier envelope implemented with COMET for the Auto-MPG dataset, and on a tree ensemble obtained with monoboost for the PimaMono dataset. (Since the running times of COMET can be significant, this experiment does not consider a comparison with the heuristic explainer Anchor (Ribeiro et al., 2018). As shown below, Anchor calls the classifier a large number of times, and that would imply unwieldy running times.)

Table 4a shows the results of running XMono using COMET as a monotonic envelope on the Auto-MPG dataset, and monoboost on the PimaMono dataset. As can be observed, the explanation sizes are in general small, which confirms the interpretability of computed AXp's and CXp's. As a general trend, CXp's are on average smaller than AXp's for Auto-MPG, but larger than AXp's for PimaMono. Moreover, the classification time completely dominates the total running time (i.e. resp. 99.99% and 99.54% of the time is spent running the classifier, independently of the classifier used). These results offer evidence that the time spent on computing explanations is in general negligible for monotonic classifiers. For both datasets, and for the instances considered, it was possible to enumerate all AXp and CXp explanations, with negligible computational overhead.

4.2 Comparison with Anchor

This section compares XMono with Anchor, using two pairs of classifiers and datasets, i.e. a monotonic decision tree for BankruptcyRisk and monoboost for PimaMono.

Table 4b shows the results of running Anchor and XMono on the BankruptcyRisk and the PimaMono datasets. XMono is significantly faster than Anchor (more than 1 order magnitude in the first case, and more than a factor of 5 in the second case). The justification is the much smaller number of calls to the classifier required by XMono than by Anchor. (While for Anchor the number of calls to the classifier can be significant, for XMono, each AXp is computed with at most a linear number of calls to the classifier. Thus, unless the number of features is very substantial, XMono has a clear performance edge over Anchor.) Somewhat surprisingly, over all instances, the average size of AXp's computed by XMono is smaller than that of Anchor for the BankruptcyRisk dataset. For the PimaMono dataset, the average size is almost the same. These results suggest that formally defined explanations need not be significantly larger than the ones computed with heuristic approaches. Furthermore, for 64.1% (resp. 18.8%) of the instances, Anchor identifies an explanation that does not hold across all points of feature space, i.e. there are points in feature space for which the explanation of Anchor holds, but the ML model makes a different prediction[14]. Observe that since XMono computes all AXp's, we can be certain about whether the explanation of Anchor is a correct explanation.

5 Conclusions & Discussion

This paper proposes novel algorithms for computing a single PI or contrastive explanation for a monotonic classifier. In contrast with earlier work (Shih et al., 2018), the complexity of the proposed algorithms is polynomial on the number of features and the time it takes the monotonic classifier to compute its predictions. As the experiments demonstrate, for simple ML models, the algorithm achieves one order of magnitude speed up when compared with a well-known heuristic explainer (Ribeiro et al., 2018), achieving better quality explanations of similar size. In contrast, for complex ML models, the experiments confirm that the running time is almost entirely spent on the classifier. Furthermore, the paper proposes a SAT-based algorithm for enumerating PI and contrastive explanations. As the experimental results show, the use of a SAT solver for enumerating PI and

[14] Similar observations have been reported elsewhere (Ignatiev, 2020).

Dataset/Tool	#Inst.	Avg. # expl.	Avg. AXp sz	Avg. CXp sz	Avg. classif. time	Avg. run time	% classif. time
AutoMPG/CMT	100	2.35	1.49	1.02	105.90s	105.92s	99.99%
PimaMono/MBT	69	9.09	1.27	3.36	16.285s	16.360s	99.54%

(a) Assessing `XMono` on the Auto-MPG and PimaMono datasets, using resp. COMET or monoboost as the classifier

Dataset	#Inst.	Anchor			`XMono` (AXp)				% diff
		Avg. Xp sz	Avg. time	# Cls calls	Avg. # Xp	Avg. Xp sz	Avg. Xp time	# Cls calls	
B. Risk	39	2.18	0.11s	1217	1.03	2.0	0.009s	24	64.1
PimaMono	69	1.26	11.2s	2967	5.64	1.27	1.8s	16	18.8

(b) Assessing `XMono` and Anchor on the Bankruptcy Risk and Pima Mono datasets

Table 4: Results of running `XMono`

contrastive explanations incurs a negligible overhead.

One possible criticism of the work is that SAT solvers are used for guiding the enumeration of explanations. This involves solving an NP-complete decision problem for each computed explanation, and so it might pose a scalability concern. (One alternative would be to consider explicit enumeration of candidate explanations, as proposed in the earlier works on model based diagnosis (Reiter, 1987; Greiner et al., 1989; Wotawa, 2001).) However, for classification problems with tens to hundreds of features and targeting thousands to tens of thousands explanations (and this far exceeds currently foreseen scenarios), the use of modern SAT reasoners (capable of solving problems with hundreds of thousands of variable and millions of clauses) can hardly be considered a limitation. Another possible criticism of this work is that full monotonicity is required. We conjecture that *full* monotonicity is necessary for tractable explanations (conditioned by the classifier run time complexity). Addressing partial monotonicity (Daniels & Velikova, 2010) is a subject of future research.

Acknowledgments. This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004, and by the H2020-ICT38 project COALA "Cognitive Assisted agile manufacturing for a Labor force supported by trustworthy Artificial intelligence".

A Additional Proofs

In the case of AXp's, Theorem 3 follows from a result on boolean monotone functions (Babin & Kuznetsov, 2011), but for clarity of exposition we opt to give a direct proof.

Theorem 3. *Determining the existence of $\lfloor N/2 \rfloor+1$ AXp's (or CXp's) of a monotonic N-feature classifier is NP-complete.*

Proof. We say that a CNF is trivially satisfiable if some literal occurs in all clauses. Clearly, SAT restricted to non-trivial CNFs is still NP-complete. Let Φ be a not trivially-satisfiable CNF on variables $x_1, \ldots, x_k$. Let $N = 2k$. Let $\tilde{\Phi}$ be identical to Φ except that each occurrence of a negative literal $\overline{x_i}$ $(1 \leq i \leq k)$ is replaced by x_{i+k}. Thus $\tilde{\Phi}$ is a CNF on N variables each of which occur only positively. Define the boolean classifier κ by $\kappa(x_1, \ldots, x_N) = 1$ if $x_i = x_{i+k} = 1$ for some $i \in \{1, \ldots, k\}$ or $\tilde{\Phi}(x_1, \ldots, x_N) = 1$ (and 0 otherwise). To show that κ is monotonic we need to show that $\mathbf{a} \leq \mathbf{b} \Rightarrow \kappa(\mathbf{a}) \leq \kappa(\mathbf{b})$. This follows by examining the two cases in which $\kappa(\mathbf{a}) = 1$: if $a_i{=}a_{i+k} \land \mathbf{a}{\leq}\mathbf{b}$, then $b_i{=}b_{i+k}$, whereas, if $\tilde{\Phi}(\mathbf{a}){=}1 \land \mathbf{a}{\leq}\mathbf{b}$, then $\tilde{\Phi}(\mathbf{b}) = 1$ (by positivity of $\tilde{\Phi}$), so in both cases $\kappa(\mathbf{b}) = 1{\geq}\kappa(\mathbf{a})$.

We first consider AXp's. Clearly $\kappa(\mathbf{1}) = 1$. There are $N/2$ obvious AXp's of this prediction, namely $(i, i{+}k)$ $(1{\leq}i{\leq}k)$. These are minimal by the assumption that Φ is not trivially satisfiable. Suppose that $\Phi(\mathbf{u}){=}1$. Let $\mathcal{X}_\mathbf{u}$ be $\{i \,|\, 1{\leq}i{\leq}k \land u_i{=}1\} \cup \{i{+}k \,|\, 1{\leq}i{\leq}k \land u_i{=}0\}$. Then (some subset of) $\mathcal{X}_\mathbf{u}$ is an AXp of the prediction $\kappa(\mathbf{1}){=}1$. The converse also holds. Thus, determining whether $\kappa(\mathbf{1}){=}1$ has more than $N/2$ AXp's is equivalent to testing the satisfiability of Φ. NP-completeness follows from the fact that $\lfloor N/2 \rfloor{+}1$ AXp's are a polytime verifiable certificate.

The proof for CXp's is similar. Clearly $\kappa(\mathbf{0}) = 0$. Again, there are $N/2$ obvious CXp's of this prediction, namely $(i, i{+}k)$ $(1{\leq}i{\leq}k)$ and (some subset of) $\mathcal{X}_\mathbf{u}$ is a CXp iff $\Phi(\mathbf{u}){=}1$. Thus, determining whether $\kappa(\mathbf{0}){=}0$ has more than $N/2$ CXp's is equivalent to testing the satisfiability of Φ, from which NP-completeness again follows. □

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A system for large-scale machine learning. In *OSDI*, pp. 265–283, 2016. Available from https://www.tensorflow.org/.

Audemard, G., Koriche, F., and Marquis, P. On tractable XAI queries based on compiled representations. In *KR*, pp. 838–849, 2020.

Audemard, G., Bellart, S., Bounia, L., Koriche, F., Lagniez, J., and Marquis, P. On the computational intelligibility of boolean classifiers. *CoRR*, abs/2104.06172, 2021. URL https://arxiv.org/abs/2104.06172.

Babin, M. A. and Kuznetsov, S. O. Enumerating minimal hypotheses and dualizing monotone boolean functions on lattices. In *FCA*, pp. 42–48, 2011.

Barile, N. and Feelders, A. Active learning with monotonicity constraints. In *SIAM ICDM*, pp. 756–767, 2012.

Bartley, C., Liu, W., and Reynolds, M. Effective monotone knowledge integration in kernel support vector machines. In *ADMA*, pp. 3–18, 2016.

Bartley, C., Liu, W., and Reynolds, M. A novel framework for constructing partially monotone rule ensembles. In *ICDE*, pp. 1320–1323, 2018.

Bartley, C., Liu, W., and Reynolds, M. Enhanced random forest algorithms for partially monotone ordinal classification. In *AAAI*, pp. 3224–3231, 2019.

Ben-David, A. Monotonicity maintenance in information-theoretic machine learning algorithms. *Mach. Learn.*, 19 (1):29–43, 1995.

Ben-David, A., Sterling, L., and Pao, Y. Learning, classification of monotonic ordinal concepts. *Comput. Intell.*, 5: 45–49, 1989.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. (eds.). *Handbook of Satisfiability*, 2009. IOS Press.

Bonakdarpour, M., Chatterjee, S., Barber, R. F., and Lafferty, J. Prediction rule reshaping. In *ICML*, pp. 629–637, 2018.

Cano, J. R., Gutiérrez, P. A., Krawczyk, B., Wozniak, M., and García, S. Monotonic classification: An overview on algorithms, performance measures and data sets. *Neurocomputing*, 341:168–182, 2019.

Chinneck, J. W. *Feasibility and Infeasibility in Optimization:: Algorithms and Computational Methods*. Springer Science & Business Media, 2008.

Choi, A., Shih, A., Goyanka, A., and Darwiche, A. On symbolically encoding the behavior of random forests. *CoRR*, abs/2007.01493, 2020. URL https://arxiv.org/abs/2007.01493.

Daniels, H. and Velikova, M. Monotone and partially monotone neural networks. *IEEE Trans. Neural Networks*, 21 (6):906–917, 2010.

Duivesteijn, W. and Feelders, A. Nearest neighbour classification with monotonicity constraints. In *ECML/PKDD*, pp. 301–316, 2008.

Fard, M. M., Canini, K. R., Cotter, A., Pfeifer, J., and Gupta, M. R. Fast and flexible monotonic functions with ensembles of lattices. In *NeurIPS*, pp. 2919–2927, 2016.

Gorji, N. and Rubin, S. Sufficient reasons for classifier decisions in the presence of constraints. *CoRR*, abs/2105.06001, 2021. URL https://arxiv.org/abs/2105.06001.

Greco, S., Matarazzo, B., and Slowinski, R. A new rough set approach to evaluation of bankruptcy risk. In *Operational tools in the management of financial risks*, pp. 121–136. Springer, 1998.

Greiner, R., Smith, B. A., and Wilkerson, R. W. A correction to the algorithm in Reiter's theory of diagnosis. *Artif. Intell.*, 41(1):79–88, 1989.

Gupta, M. R., Cotter, A., Pfeifer, J., Voevodski, K., Canini, K. R., Mangylov, A., Moczydlowski, W., and Esbroeck, A. V. Monotonic calibrated interpolated look-up tables. *J. Mach. Learn. Res.*, 17:109:1–109:47, 2016.

Huang, X., Izza, Y., Ignatiev, A., and Marques-Silva, J. On efficiently explaining graph-based classifiers. *CoRR*, abs/2106.01350, 2021. URL https://arxiv.org/abs/2106.01350.

Ignatiev, A. Towards trustable explainable AI. In *IJCAI*, pp. 5154–5158, 2020.

Ignatiev, A. and Marques-Silva, J. SAT-based rigorous explanations for decision lists. *CoRR*, abs/2105.06782, 2021. URL https://arxiv.org/abs/2105.06782.

Ignatiev, A., Narodytska, N., and Marques-Silva, J. Abduction-based explanations for machine learning models. In *AAAI*, pp. 1511–1519, 2019.

Ignatiev, A., Narodytska, N., Asher, N., and Marques-Silva, J. On relating 'why?' and 'why not?' explanations. *CoRR*, abs/2012.11067, 2020. URL https://arxiv.org/abs/2012.11067.

Izza, Y. and Marques-Silva, J. On explaining random forests with SAT. *CoRR*, abs/2105.10278, 2021. URL https://arxiv.org/abs/2105.10278.

Izza, Y., Ignatiev, A., and Marques-Silva, J. On explaining decision trees. *CoRR*, abs/2010.11034, 2020. URL https://arxiv.org/abs/2010.11034.

Izza, Y., Ignatiev, A., Narodytska, N., Cooper, M. C., and Marques-Silva, J. Efficient explanations with relevant sets. *CoRR*, abs/2106.00546, 2021. URL https://arxiv.org/abs/2106.00546.

Liffiton, M. H., Previti, A., Malik, A., and Marques-Silva, J. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016.

Liu, X., Han, X., Zhang, N., and Liu, Q. Certified monotonic neural networks. In *NeurIPS*, 2020.

Lundberg, S. M. and Lee, S. A unified approach to interpreting model predictions. In *NeurIPS*, pp. 4765–4774, 2017.

Magdon-Ismail, M. and Sill, J. A linear fit gets the correct monotonicity directions. *Mach. Learn.*, 70(1):21–43, 2008.

Marques-Silva, J., Gerspacher, T., Cooper, M. C., Ignatiev, A., and Narodytska, N. Explaining naive bayes and other linear classifiers with polynomial time and delay. In *NeurIPS*, 2020.

Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

Potharst, R. and Bioch, J. C. Decision trees for ordinal classification. *Intell. Data Anal.*, 4(2):97–111, 2000.

Reiter, R. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

Ribeiro, M. T., Singh, S., and Guestrin, C. "Why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pp. 1135–1144, 2016.

Ribeiro, M. T., Singh, S., and Guestrin, C. Anchors: High-precision model-agnostic explanations. In *AAAI*, pp. 1527–1535, 2018.

Shi, W., Shih, A., Darwiche, A., and Choi, A. On tractable representations of binary neural networks. In *KR*, pp. 882–892, 2020.

Shih, A., Choi, A., and Darwiche, A. A symbolic approach to explaining bayesian network classifiers. In *IJCAI*, pp. 5103–5111, 2018.

Sill, J. Monotonic networks. In *NIPS*, pp. 661–667, 1997.

Sivaraman, A., Farnadi, G., Millstein, T. D., and den Broeck, G. V. Counterexample-guided learning of monotonic neural networks. In *NeurIPS*, 2020.

Van den Broeck, G., Lykov, A., Schleich, M., and Suciu, D. On the tractability of SHAP explanations. *CoRR*, abs/2009.08634, 2020. URL https://arxiv.org/abs/2009.08634.

van der Gaag, L. C., Bodlaender, H. L., and Feelders, A. J. Monotonicity in bayesian networks. In *UAI*, pp. 569–576, 2004.

Verbeke, W., Martens, D., and Baesens, B. RULEM: A novel heuristic rule learning approach for ordinal classification with monotonicity constraints. *Appl. Soft Comput.*, 60:858–873, 2017.

Wäldchen, S., MacDonald, J., Hauch, S., and Kutyniok, G. The computational complexity of understanding binary classifier decisions. *J. Artif. Intell. Res.*, 70:351–387, 2021. doi: 10.1613/jair.1.12359. URL https://doi.org/10.1613/jair.1.12359.

Wang, S. and Gupta, M. R. Deontological ethics by monotonicity shape constraints. In *AISTATS*, pp. 2043–2054, 2020.

Wotawa, F. A variant of Reiter's hitting-set algorithm. *Inf. Process. Lett.*, 79(1):45–51, 2001.

You, S., Ding, D., Canini, K. R., Pfeifer, J., and Gupta, M. R. Deep lattice networks and partial monotonic functions. In *NeurIPS*, pp. 2981–2989, 2017.