Answer-Set Programs for Reasoning about Counterfactual Interventions and Responsibility Scores for Classification

Leopoldo Bertossi[1,2] and **Gabriela Reyes**[1]

Universidad Adolfo Ibáñez[1]
Faculty of Engineering and Sciences
and
Millennium Inst. for Foundational Research on Data (IMFD)[2]
Santiago, Chile
leopoldo.bertossi@uai.cl and gabreyes@alumnos.uai.cl

Abstract. We describe how *answer-set programs* can be used to declaratively specify counterfactual interventions on entities under classification, and reason about them. In particular, they can be used to define and compute responsibility scores as attribution-based explanations for outcomes from classification models. The approach allows for the inclusion of domain knowledge and supports query answering. A detailed example with a naive-Bayes classifier is presented.

1 Introduction

Counterfactuals are at the very basis of the notion of *actual causality* [18]. They are hypothetical interventions (or changes) on variables that are part of a causal structural model. Counterfactuals can be used to define and assign *responsibility scores* to the variables in the model, with the purpose of quantifying their causal contribution strength to a particular outcome [12, 19]. These generals notions of actual causality have been successfully applied in databases, to investigate actual causes and responsibilities for query results [25, 26, 2].

Numerical scores have been applied in *explainable AI*, and most prominently with machine learning models for classification [28]. Usually, feature values in entities under classification are given numerical scores, to indicate how relevant those values are for the outcome of the classification. For example, one might want to know how important is the city or the neighborhood where a client lives when a bank uses a classification algorithm to accept or reject his/her loan request. We could, for example, obtain a large responsibility score for the feature value "Bronx in New York City". As such it is a *local explanation*, for the entity at hand, and in relation to its participating feature values.

A widely used score is Shap [24], that is based on the Shapley value of coalition game theory [30]. As such, it is based on *implicit* counterfactuals and a numerical aggregation of the outcomes from classification for those different counterfactual versions of the initial entity. Accordingly, the emphasis is not on the possible counterfactuals, but on the final numerical score. However, counterfactuals are interesting *per se*. For example, we might want to know if the client, by changing his/her address, might turn a rejection into the acceptance of the loan request. The so generated new entity, with a new address and a new label, is a *counterfactual version* of the original entity.

In [5] the x-Resp score was introduced. It is defined in terms of explicit counterfactuals and responsibility as found in general actual causality. A more general version of it, the Resp score, was introduced in [3], and was compared with other scores, among them, Shap. For simplicity we will concentrate on x-Resp.

Following up our interest in counterfactuals, we propose *counterfactual intervention programs* (CIPs). They are *answer-set programs* (ASPs) [9, 16] that are used to specify counterfactual versions of an initial entity, and compute the x-Resp scores for its feature values. More specifically, here we present approaches to- and results about the use of ASPs for specifying counterfactual interventions on entities under classification, and reasoning about them. In this work, we show CIPs and their use in the light of a naive-Bayes classifier. See [5] for more details and an example with a decision-tree classifier; and [6] for more examples of the use of ASPs for actual causality and responsibility.

ASP is a flexible and powerful logic programming paradigm that, as such, allows for declarative specifications and reasoning from them. The (non-monotonic) semantics of a program is given in terms of its *stable models*, i.e. special models that make the program true [15]. In our applications, the relevant counterfactual versions correspond to different models of the CIP. In our example with a naive-Bayes classifier, we use the *DLV* system [23] and its *DLV-Complex* extension [10, 11] that implement the ASP semantics; the latter with set- and numerical aggregations.

CIPs can be used to specify the relevant counterfactuals, analyze different versions of them, and use them to specify and compute the x-Resp score. By using additional features of ASP, and of *DLV* in particular, for example *strong and weak program constraints*, one can specify and compute maximum-responsibility counterfactuals. The classifier can be specified directly in the CIP, or can be invoked as an external predicate [5]. The latter case could be that of a *black-box classifier* [31], to which Shap and x-Resp can be applied.

CIPs are very flexible in that one can easily add *domain knowledge* or *domain semantics*, in such a way that certain counterfactuals are not considered, or others are privileged. In particular, one can specify *actionable counterfactuals*, that, in certain applications, make more sense and may lead to feasible changes of feature values for an entity to reverse a classification result [32, 21]. All these changes are much more difficult to implement if we use a purely procedural approach. With CIPs, many changes of potential interest can be easily and seamlessly tried out on-the-fly, for exploration purposes.

Reasoning is enabled by query answering, for which two semantics are offered. Under the *brave semantics* one obtains as query answers those that hold in *some* model of the CIP. This can be useful to detect if there is "minimally changed" counterfactual version of the initial entity where the city is changed together with the salary. Under the *cautious semantics* one obtains answers that hold in all the models of the CIP, which could be used to identify feature values that do not change no matter what when we reverse the outcome. Query answering on ASPs offers many opportunities.

This paper is structured as follows. In Section 2, we introduce and discuss the problem, and provide an example. In Section 3 we introduce the naive-Bayes classifier we will use as a running example. In Section 4 we define the x-Resp score. In Section 5 we introduce *counterfactual intervention programs*. In Section 6, we discuss the use of domain knowledge and query answering. We end in Section 7 with some final conclusions. An extended version of this paper can be found in [7], which in its Appendix A provides the basics of answer-set programming; and in its Appendix B presents the complete program for the running example, in *DLV* code, and its output.

2 Counterfactual Interventions and Explanation Scores

We consider a finite set of features, $\mathcal{F}$, with each feature $F \in \mathcal{F}$ having a finite domain, $Dom(F)$, where F, as a function, takes its values. The features are applied to entities $\mathbf{e}$ that belong to a population $\mathcal{E}$. Actually, we identify the entity $\mathbf{e}$ with the record (or tuple) formed by the values the features take on it: $\mathbf{e} = \langle F_1(\mathbf{e}), \ldots, F_n(\mathbf{e})\rangle$. Now, entities in $\mathcal{E}$ go through a *classifier*, C, that returns *labels* for them. We will assume the classifier is binary, e.g. the labels could be 1 or 0.

In Figure 1, we have a classifier receiving as input an entity, $\mathbf{e}$. It returns as an output a label, $L(\mathbf{e})$, corresponding to the classification of input $\mathbf{e}$. In principle, we could see $\mathcal{C}$ as a black-box, in the sense that only by direct interaction with it, we have access to its input/output relation. That is, we may have no access to the mathematical classification model inside $\mathcal{C}$.

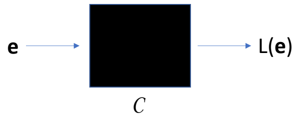


Fig. 1. A black-box classifier

The entity $\mathbf{e}$ could represent a client requesting a loan from a financial institution. The classifier of the latter, on the basis of $\mathbf{e}$'s feature values (e.g. for EdLevel, Income, Age, etc.) assigns the label 1, for rejection. An explanation may be requested by the client. Explanations like this could be expected from any kind of classifier. It could be an explicit classification model, e.g. a classification tree or a logistic regression model. In these cases, we might be in a better position to give an explanation, because we can inspect the internals of the model [31]. However, we can find ourselves in the "worst-case scenario" in which we do not have access to the internal model. That is, we are confronted to a black-box classifier, and we still have to provide explanations.

An approach to explanations that has become popular, specially in the absence of the model, assigns numerical *scores* to the feature values for an entity, trying to answer the question about which of the feature values contribute the most to the received label.

Example 1. We reuse a popular example from [27]. The set of features is $\mathcal{F} = \{\mathsf{Outlook}, \mathsf{Temperature}, \mathsf{Humidity}, \mathsf{Wind}\}$, with $\mathit{Dom}(\mathsf{Outlook}) = \{\mathsf{sunny}, \mathsf{overcast}, \mathsf{rain}\}$, $\mathit{Dom}(\mathsf{Temperature}) = \{\mathsf{high}, \mathsf{medium}, \mathsf{low}\}$, $\mathit{Dom}(\mathsf{Humidity}) = \{\mathsf{high}, \mathsf{normal}\}$, $\mathit{Dom}(\mathsf{Wind}) = \{\mathsf{strong},\ \mathsf{weak}\}$. We will always use this order for the features.

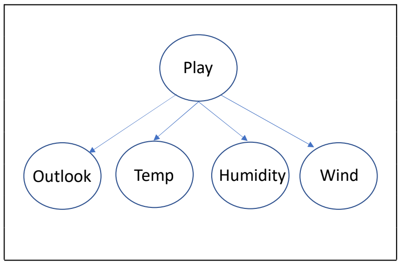
Now, assume we have a classifier, $\mathcal{C}$, that allows us to decide if we play tennis (label yes) or not (label no) under a given combination of weather features. A concrete *naive-Bayes classifier* will be given in Section 3. For example, a particular weather entity has a value for each of the features, e.g. $\mathbf{e} = \mathit{ent}(\mathsf{rain}, \mathsf{high}, \mathsf{normal}, \mathsf{weak})$. We want to decide about playing tennis or not under the wether conditions represented by $\mathbf{e}$. □

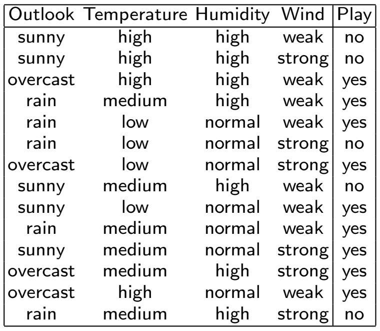
Score-based methodologies are sometimes based on *counterfactual interventions*: *What would happen with the label if we change this particular value, leaving the others fixed?* Or the other way around: *What if we leave this value fixed, and change the others?* The resulting labels from these counterfactual interventions can be aggregated in different ways, leading to a score for the feature value under inspection.

Let us illustrate these questions by using the entity $\mathbf{e}$ in the preceding example. If we use the *naive-Bayes classifier* with entity $\mathbf{e}$, we obtain the label yes (c.f. Section 3). In order to detect and quantify the relevance (technically, the responsibility) of a feature value in $\mathbf{e} = \mathit{ent}(\mathsf{rain}, \mathsf{high}, \underline{\mathsf{normal}}, \mathsf{weak})$, say, of feature Humidity (underlined), we *hypothetically intervene* its value. In this case, if we change it from normal to high, we obtain a new entity $\mathbf{e}' = \mathit{ent}(\mathsf{rain}, \mathsf{high}, \mathsf{high}, \mathsf{weak})$. If we input this entity $\mathbf{e}'$ into the classifier, we now obtain the label no. We say that $\mathbf{e}'$ is a *counterfactual version* of $\mathbf{e}$.

This change of label is an indication that the original feature value for Humidity is indeed relevant for the original classification. Furthermore, the fact that it is good enough to change only this individual value is an indication of its strength. If, to change the label, we also had to change other values together with that for Humidity, its strength would be lower. In Section 4, we revisit a particular *responsibility score*, x-Resp, which captures this intuition, and can be applied with black-box or open models.

3 A Naive-Bayes Classifier





Outlook	Temperature	Humidity	Wind	Play
sunny	high	high	weak	no
sunny	high	high	strong	no
overcast	high	high	weak	yes
rain	medium	high	weak	yes
rain	low	normal	weak	yes
rain	low	normal	strong	no
overcast	low	normal	strong	yes
sunny	medium	high	weak	no
sunny	low	normal	weak	yes
rain	medium	normal	weak	yes
sunny	medium	normal	strong	yes
overcast	medium	high	strong	yes
overcast	high	normal	weak	yes
rain	medium	high	strong	no

Example 2. (example 1 cont.) We now we build a naive-Bayes classifier for the binary variable Play, about playing tennis or not. A Bayesian network, that is the basis for this classifier, is shown right here above (left). In addition to the network structure, we have to assign probability distributions to the nodes in it. These distributions are learned from the training data in the table (right).

In this case, the features stochastically depend on the output variable Play, and are independent from each other given the output. To fully specify the network, we need the absolute distribution for the top node; and the conditional distributions for the lower nodes.

These are the distributions inferred from the frequencies in the training data:

$P(\mathsf{Play} = \mathsf{yes}) = \frac{9}{14}$	$P(\mathsf{Play} = \mathsf{no}) = \frac{5}{14}$
$P(\mathsf{Outlook} = \mathsf{sunny} \vert \mathsf{Play} = \mathsf{yes}) = \frac{2}{9}$	$P(\mathsf{Outlook} = \mathsf{sunny} \vert \mathsf{Play} = \mathsf{no}) = \frac{3}{5}$
$P(\mathsf{Outlook} = \mathsf{overcast} \vert \mathsf{Play} = \mathsf{yes}) = \frac{4}{9}$	$P(\mathsf{Outlook} = \mathsf{overcast} \vert \mathsf{Play} = \mathsf{no}) = 0$
$P(\mathsf{Outlook} = \mathsf{rain} \vert \mathsf{Play} = \mathsf{yes}) = \frac{3}{9}$	$P(\mathsf{Outlook} = \mathsf{rain} \vert \mathsf{Play} = \mathsf{no}) = \frac{2}{5}$
$P(\mathsf{Temp} = \mathsf{high} \vert \mathsf{Play} = \mathsf{yes}) = \frac{2}{9}$	$P(\mathsf{Temp} = \mathsf{high} \vert \mathsf{Play} = \mathsf{no}) = \frac{2}{5}$
$P(\mathsf{Temp} = \mathsf{medium} \vert \mathsf{Play} = \mathsf{yes}) = \frac{4}{9}$	$P(\mathsf{Temp} = \mathsf{medium} \vert \mathsf{Play} = \mathsf{no}) = \frac{2}{5}$
$P(\mathsf{Temp} = \mathsf{low} \vert \mathsf{Play} = \mathsf{yes}) = \frac{3}{9}$	$P(\mathsf{Temp} = \mathsf{low} \vert \mathsf{Play} = \mathsf{no}) = \frac{1}{5}$
$P(\mathsf{Humidity} = \mathsf{high} \vert \mathsf{Play} = \mathsf{yes}) = \frac{3}{9}$	$P(\mathsf{Humidity} = \mathsf{high} \vert \mathsf{Play} = \mathsf{no}) = \frac{4}{5}$
$P(\mathsf{Humidity} = \mathsf{normal} \vert \mathsf{Play} = \mathsf{yes}) = \frac{6}{9}$	$P(\mathsf{Humidity} = \mathsf{normal} \vert \mathsf{Play} = \mathsf{no}) = \frac{1}{5}$
$P(\mathsf{Wind} = \mathsf{strong} \vert \mathsf{Play} = \mathsf{yes}) = \frac{3}{9}$	$P(\mathsf{Wind} = \mathsf{strong} \vert \mathsf{Play} = \mathsf{no}) = \frac{3}{5}$
$P(\mathsf{Wind} = \mathsf{weak} \vert \mathsf{Play} = \mathsf{yes}) = \frac{6}{9}$	$P(\mathsf{Wind} = \mathsf{weak} \vert \mathsf{Play} = \mathsf{no}) = \frac{2}{5}$

We can use them to decide, for example, about playing or not with the following input data: Outlook = rain, Temp = high, Humidity = normal, Wind = weak. If we keep this order of the features, we are classifying the weather entity $\mathbf{e} = \langle \mathsf{rain}, \mathsf{high}, \mathsf{normal}, \mathsf{weak} \rangle$. This is done by determining the maximum probability between the two probabilities:

$$P\big(\mathsf{Play} = \mathsf{yes} \big| \mathsf{Outlook} = \mathsf{rain}, \mathsf{Temp} = \mathsf{high}, \mathsf{Humidity} = \mathsf{normal}, \mathsf{Wind} = \mathsf{weak}\big), \tag{1}$$

$$P\big(\mathsf{Play} = \mathsf{no} \big| \mathsf{Outlook} = \mathsf{rain}, \mathsf{Temp} = \mathsf{high}, \mathsf{Humidity} = \mathsf{normal}, \mathsf{Wind} = \mathsf{weak}\big). \tag{2}$$

Now, for each of the probabilities of the form $P(\mathsf{P}|\mathsf{O}, \mathsf{T}, \mathsf{H}, \mathsf{W})$ it holds:

$$P(\mathsf{P}|\mathsf{O}, \mathsf{T}, \mathsf{H}, \mathsf{W}) = \frac{P(\mathsf{P}, \mathsf{O}, \mathsf{T}, \mathsf{H}, \mathsf{W})}{P(\mathsf{O}, \mathsf{T}, \mathsf{H}, \mathsf{W})} = \frac{P(\mathsf{O}|\mathsf{P})P(\mathsf{T}|\mathsf{P})P(\mathsf{H}|\mathsf{P})P(\mathsf{W}|\mathsf{P})P(\mathsf{P})}{\sum_{\mathsf{P}} P(\mathsf{O}|\mathsf{P})P(\mathsf{T}|\mathsf{P})P(\mathsf{H}|\mathsf{P})P(\mathsf{W}|\mathsf{P})P(\mathsf{P})}. \tag{3}$$

In particular, the numerators for (1) and (2) become, resp.:

$$\begin{aligned} &P\big(\mathsf{Outlook} = \mathsf{rain} \big| \mathsf{Play} = \mathsf{yes}\big) P\big(\mathsf{Temp} = \mathsf{high} \big| \mathsf{Play} = \mathsf{yes}\big) P\big(\mathsf{Humidity} = \mathsf{normal} \big| \mathsf{Play} = \mathsf{yes}\big) \times \\ &\times P\big(\mathsf{Wind} = \mathsf{false} \big| \mathsf{Play} = \mathsf{yes}\big) P\big(\mathsf{Play} = \mathsf{yes}\big) = \frac{3}{9}\frac{2}{9}\frac{6}{9}\frac{6}{9}\frac{9}{14} = \frac{4}{189}, \qquad (4) \\ &P\big(\mathsf{Outlook} = \mathsf{rain} \big| \mathsf{Play} = \mathsf{no}\big) P\big(\mathsf{Temp} = \mathsf{high} \big| \mathsf{Play} = \mathsf{no}\big) P\big(\mathsf{Humidity} = \mathsf{normal} \big| \mathsf{Play} = \mathsf{no}\big) \times \\ &\times P\big(\mathsf{Wind} = \mathsf{false} \big| \mathsf{Play} = \mathsf{no}\big) P\big(\mathsf{Play} = \mathsf{no}\big) = \frac{2}{5}\frac{2}{5}\frac{1}{5}\frac{2}{5}\frac{5}{14} = \frac{4}{875}. \qquad (5) \end{aligned}$$

The denominator for both cases is the marginal probability, i.e. $\frac{4}{189} + \frac{4}{875}$. Then, it is good enough to compare (4) and (5). Since the former is larger, the decision (or classification) becomes: Play = yes. □

4 The x-Resp Score

Assume that an entity $\mathbf{e}$ has received the label 1 by the classifier $\mathcal{C}$, and we want to explain this outcome by assigning numerical scores to $\mathbf{e}$'s feature values, in such a way, that a higher score for a feature value reflects that it has been important for the outcome. We do this now using the x-Resp score, whose definition we motivate below by means of an example. The x-Resp score as defined below is not restricted to- but more suitable for binary features, i.e. that take the values true or false (or 1 and 0, resp.). The generalization in [5] is more appropriate for multi-valued features. C.f. Section 7 for a discussion, and [4, 5] for more details.

Example 3. In Figure 2, the black box is classifier $\mathcal{C}$. An entity $\mathbf{e}$ has gone through it obtaining label 1, shown in the first row in the figure. We want to assign a score to the feature value $\mathbf{x}$ for a feature $F \in \mathcal{F}$. We proceed, counterfactually, changing the value $\mathbf{x}$ into $\mathbf{x}'$, obtaining a counterfactual version $\mathbf{e}_1$ of $\mathbf{e}$. We classify $\mathbf{e}_1$, and we still get the outcome 1 (second row). In between, we may counterfactually change other feature values, $\mathbf{y}, \mathbf{z}$ in $\mathbf{e}$, into $\mathbf{y}', \mathbf{z}'$, but keeping $\mathbf{x}$, obtaining entity $\mathbf{e}_2$, and the outcome does not change (third row). However, if we change in $\mathbf{e}_2$, $\mathbf{x}$ into $\mathbf{x}'$, the outcome does change (fourth row).

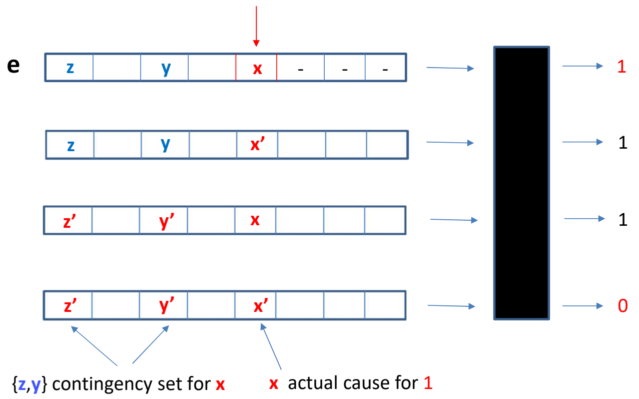


Fig. 2. Classified entity and its counterfactual versions

This shows that the value $\mathbf{x}$ is relevant for the original output, but, for this outcome, it needs company, say of the feature values $\mathbf{y}, \mathbf{z}$ in $\mathbf{e}$. According to *actual causality*, we can say that the feature value $\mathbf{x}$ in $\mathbf{e}$ is an *actual cause* for the classification, that needs a *contingency set* formed by the values $\mathbf{y}, \mathbf{z}$ in $\mathbf{e}$. In this case, the contingency set has size 2. If we found a contingency set for $\mathbf{x}$ of size 1 in $\mathbf{e}$, we would consider $\mathbf{x}$ even more relevant for the output. $\square$

On this basis, we can define [4, 5], for a feature value $\mathbf{x}$ in $\mathbf{e}$: (a) $\mathbf{x}$ is a *counterfactual explanation* for $L(\mathbf{e}) = 1$ if $L(\mathbf{e}\frac{\mathbf{x}}{\mathbf{x}'}) = 0$, for some $\mathbf{x}' \in \mathit{Dom}(F)$ (the domain of feature F). (Here we use the common notation $\mathbf{e}\frac{\mathbf{x}}{\mathbf{x}'}$ for the entity obtained by replacing $\mathbf{x}$ by $\mathbf{x}'$ in $\mathbf{e}$.) (b) $\mathbf{x}$ is an *actual explanation* for $L(\mathbf{e}) = 1$ if there is a contingency set of values $\mathbf{Y}$ in $\mathbf{e}$, with $\mathbf{x} \notin \mathbf{Y}$, and new values $\mathbf{Y}' \cup \{\mathbf{x}'\}$, such that $L(\mathbf{e}\frac{\mathbf{Y}}{\mathbf{Y}'}) = 1$ and $L(\mathbf{e}\frac{\mathbf{x}\mathbf{Y}}{\mathbf{x}'\mathbf{Y}'}) = 0$. We say that $\mathbf{Y}$ is *minimal* if there is no $\mathbf{Y}'$ with $\mathbf{Y}' \subsetneqq \mathbf{Y}$, that is also a contingency set for $\mathbf{x}$ in $\mathbf{e}$. Similarly, $\mathbf{Y}$ is *miminum* contingency set if it is a a minimum-size contingency set for $\mathbf{x}$ in $\mathbf{e}$.

Contingency sets may come in sizes from 0 to $n-1$ for feature values in records of length n. Accordingly, we can define for the actual cause $\mathbf{x}$ in $\mathbf{e}$: If $\mathbf{Y}$ is a minimum contingency set for $\mathbf{x}$, $\textsf{x-Resp}(\mathbf{e}, \mathbf{x}) := \frac{1}{1+|\mathbf{Y}|}$; and as 0 when $\mathbf{x}$ is not an actual cause. (C.f. Section 7 for the Resp score that generalizes x-Resp.)

We will reserve the notion of counterfactual explanation *for (or* counterfactual version *of) an input entity* $\mathbf{e}$ *for any entity* $\mathbf{e}'$ *obtained from* $\mathbf{e}$ *by modifying feature values in* $\mathbf{e}$ *and that leads to a different label, i.e.* $L(\mathbf{e}) \neq L(\mathbf{e}')$*. Notice that from such an* $\mathbf{e}'$ *we can read off actual causes for* $L(\mathbf{e})$ *as feature values, and contingency sets for those actual causes. It suffices to compare* $\mathbf{e}$ *with* $\mathbf{e}'$*.*

In Section 5 we give a detailed example that illustrates these notions, and also shows the use of ASPs for the specification and computation of counterfactual versions of a given entity, and the latter's x-Resp score.

5 Counterfactual-Intervention Programs

Together with illustrating the notions introduced in Section 4, we will introduce, by means of an example, *Counterfactual Intervention Programs* (CIPs). The program corresponds to the naive-Bayes classifier presented in Section 3.

CIPs are *answer-set programs* (ASPs) that specify the counterfactual versions of a given entity, and also, if so desired, only the *maximum-responsibility* counterfactual explanations, i.e. counterfactual versions that lead to a maximum x-Resp score. (C.f. [5] for many more details and examples with *decision trees* as classifiers.).

Example 4. (examples 1 and 2 cont.) We present now the CIP in *DLV-Complex* notation. Since the program specifies and applies counterfactual changes of attribute values, we have to indicate when an intervention is applied, when an entity may still be subject to additional interventions, and when a final version of an entity has been reached, i.e. the label has been changed. To achieve this, the program uses annotation constants `o`, for "original entity", `do`, for "do a counterfactual intervention" (a single change of feature value), `tr`, for "entity in transition", and `s`, for "stop, the label has changed".

We will explain the program along the way, as we present it, and with additional explanations as comments written directly in the *DLV* code. We will keep the most relevant parts of the program. The complete program can be found in [7, Appendix B].

The absolute and conditional probabilities will be given as facts of the *DLV* program. They are represented as percentages, because *DLV* handles operations with integer numbers. The conditional probabilities are atoms of the form `p_f_c(feature value, play outcome, prob\%)`, with "`f`" suggesting the feature name, and "`c`", that it is a conditional probability. For example, `p_h_c(normal, yes, 67)` is the conditional probability (of 67%) of Humidity being normal given that Play takes value yes. Similarly, this is an absolute probability for Play: `p(yes, 64)`.

The program has as facts also the contents of the domains. They are of the form dom_f(feature value), with "f" suggesting the feature name again, e.g. dom_h(high), for Humidity. Finally, among the facts we find the original entity that will be intervened by means of the CIP. In this case, as in Example 1, ent(e,rain,high,normal,weak,o), where constant e is an entity identifier (eid), and o is the annotation constant. This entity gets label yes, i.e. Play = yes. Through interventions, we expect the label to become no, i.e. Play = no.

Aggregation functions over sets will be needed later in the program, to build *contingency sets* (c.f. Section 4). So, we use *DLV-Complex* that supports this functionality. "List and Sets" has to be specified at the beginning of the program, together with the maximum integer value. This is the first part of the CIP, showing the facts: (as usual, words starting with lower case are constants; whereas with upper case, variables)

```
% DLV-COMPLEX    #include<ListAndSet>   #maxint = 100000000.
% domains:
   dom_o(sunny). dom_o(overcast). dom_o(rain). dom_t(high). dom_t(medium).
   dom_t(low). dom_h(high). dom_h(normal). dom_w(strong). dom_w(weak).
% original entity that gets label 1:
   ent(e,rain,high,normal,weak,o).
% absolute probabilities for Play (as percentage)
   p(yes, 64). p(no, 36).
% Outlook conditional probabilities (as percentage)
   p_o_c(sunny, yes, 22). p_o_c(overcast, yes, 45). p_o_c(rain, yes, 33).
   p_o_c(sunny, no, 60). p_o_c(overcast, no, 0). p_o_c(rain, no, 40).
% Temperature conditional probabilities (as percentage)
   p_t_c(high, yes, 22). p_t_c(medium, yes, 45). p_t_c(low, yes, 33).
   p_t_c(high, no, 40). p_t_c(medium, no, 40). p_t_c(low, no, 20).
% Humidity conditional probabilities (as percentage)
   p_h_c(normal, yes, 67). p_h_c(high, yes, 33).
   p_h_c(normal, no, 20). p_h_c(high, no, 80).
% Wind conditional probabilities (as percentage)
   p_w_c(strong, yes, 33). p_w_c(weak, yes, 67).
   p_w_c(strong, no, 60). p_w_c(weak, no, 40).
```

The classifier will compute posterior probabilities for Play according to equations (1) and (2) in Section 3. Next, they are compared, and the largest determines the label. As we can see from equation (3), the denominator is irrelevant for this comparison. So, we need only the numerators. They are specified by means of a predicate of the form pb_num(E,O,T,H,W,V,Fp), where the arguments stand for: eid, (values for) Outlook, Temp, Humidity, Wind and Play, resp.; and the probability as a percentage. The CIP has to specify predicate pb_num(E,O,T,H,W,V,Fp). That part of the program is not particularly interesting, and looks somewhat cumbersome due to the combination of simple arithmetical operations with probabilities. C.f. the program in [7, Appendix B].

Next, we have to specify the transition annotation constant tr, that is used in rule bodies below. It indicates that we are using an entity that is in transition. This annotation is specified as follows:

```
% transition rules: the initial entity or one affected by an intervention
  ent(E,O,T,H,W,tr) :- ent(E,O,T,H,W,o).
  ent(E,O,T,H,W,tr) :- ent(E,O,T,H,W,do).
```

Now we have to specify the classifier, or better, the classification criteria, appealing to predicate pb_num(E, O, H, W, V, Fp). More precisely, we have to compare Fp for Play value yes, denoted Fyes, with Fp for Play value no, denoted Fno. If the former is larger, we obtain label yes; otherwise label no:

```
% spec of the classifier
  cls(E,O,T,H,W,yes) :- ent(E,O,T,H,W,tr),  pb_num(E,O,T,H,W,yes,Fyes),
                        pb_num(E,O,T,H,W,no,Fno), Fyes >= Fno.
  cls(E,O,T,H,W,no)  :- ent(E,O,T,H,W,tr),  pb_num(E,O,T,H,W,yes,Fyes),
                        pb_num(E,O,T,H,W,no,Fno), Fyes < Fno.
```

Notice the use of annotation constant tr in the body, because we will be classifying entities that are in transition. Next, the CIP specifies all the one-step admissible counterfactual interventions on entities with label yes, which produces entities in transition. This disjunctive rule is the main rule.

```
% counterfactual rule: alternative single-value changes
  ent(E,Op,T,H,W,do) v ent(E,O,Tp,H,W,do) v
  ent(E,O,T,Hp,W,do) v ent(E,O,T,H,Wp,do) :- ent(E,O,T,H,W,tr),
       cls(E,O,T,H,W,yes), O != Op, T != Tp, H!= Hp, W!= Wp,
       chosen_o(O,T,H,W,Op), chosen_t(O,T,H,W,Tp), chosen_h(O,T,H,W,Hp),
       chosen_w(O,T,H,W,Wp), dom_o(Op), dom_t(Tp), dom_h(Hp), dom_w(Wp).
```

Here we are using predicates chosen, one for each of the four features. For example, chosen_h(O,T,H,W,Hp) "chooses" for each combination of values, O,T,H,W for Outlook, Temp, Humidity, and Wind, a unique (and new) value Hp for feature Humidity, and that value is taken from its domain dom_h. Through an intervention, that value Hp replaces the original value H, as one of the four possible value changes that are indicated in the rule head.

The semantics of ASPs makes only one of the possible disjuncts in the head true (unless forced otherwise by other rules in the program, which does not happen with CIPs). The chosen predicates can be specified in a generic manner [17]. Here, we skip their specification, but they can be found in [7, Appendix B].

In order to avoid going back to the original entity through counterfactual interventions, we may impose a *hard program constraint* [23]. These constraints are rules with empty head, which capture a negation. They have the effect of discarding the models where the body becomes true. In this case:

```
% not going back to initial entity
      :- ent(E,O,T,H,W,do), ent(E,O,T,H,W,o).
```

Next, we stop performing interventions when we switch the label to no, which introduces the annotation s:

```
% stop when label has been changed:
  ent(E,O,T,H,W,s) :- ent(E,O,T,H,W,do), cls(E,O,T,H,W,no).
```

Finally, we introduce an extra program constraint, to avoid computing models where the original entity never changes label. Those models will not contain the original eid with annotation `s`:

```
% extra constraint avoiding models where label does not change
    :- ent(E,O,T,H,W,o), not entAux(E).
% auxiliary predicate to avoid unsafe negation right above
  entAux(E) :- ent(E,O,T,H,W,s).
```

The rest of the program uses counterfactual interventions to collect individual changes (next rules), sets of them, cardinalities of those sets, etc.

```
% collecting changed values for each feature:
  expl(E,outlook,O)  :- ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), O != Op.
  expl(E,temp,T)     :- ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), T != Tp.
  expl(E,humidity,H) :- ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), H != Hp.
  expl(E,wind,W)     :- ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), W != Wp.
```

With them, we will obtain, for example, the atom `expl(e,humidity,normal)` in some of the models of the program, because there is a counterfactual entity that changes normal humidity into high (c.f. Section 2). The atom indicates that original value `normal` for `humidity` is part of an explanation for entity `e`. Contingency sets for a feature value are obtained with the rules below. Since we keep everywhere the eid, it is good enough to collect the names of the features whose values are changed. For this we use predicate `cont(E,U,S)`. Here, `U` is a feature (with changed value), `S` is the *set of all* feature names whose values are changed together with that for `U`. These sets are build using the built-in set functions of *DLV-Complex*. Similarly with the built-in set membership check.

```
% building  contingency sets
  cause(E,U)       :- expl(E,U,X).
  cauCont(E,U,I)   :- expl(E,U,X), expl(E,I,Z), U != I.
  preCont(E,U,{I}) :- cauCont(E,U,I).
  preCont(E,U,#union(Co,{I})) :- cauCont(E,U,I), preCont(E,U,Co),
                                 not #member(I,Co).
  cont(E,U,Co)     :- preCont(E,U,Co), not HoleIn(E,U,Co).
  HoleIn(E,U,Co)   :- preCont(E,U,Co), cauCont(E,U,I), not #member(I,Co).
  tmpCont(E,U)     :- cont(E,U,Co), not #card(Co,0).
  cont(E,U,{})     :- cause(E,U), not tmpCont(E,U).
```

The construction is such that one keeps adding contingency features, using pre-contingency sets, until there is nothing else to add. In this way the contingency sets contain all the features that have to be changed with the one at hand `U`. For example, in one of the models we will find the atom `cont(e,humidity,{})`, meaning that a change of the humidity value alone, i.e. with empty contingency set, suffices to switch the label. Each counterfactual version of entity $\mathbf{e}$ will be represented by a model of the program. Due to model minimality, the associated set of changes of feature values that accompany a counterfactual change of feature value, say $\mathbf{x}$ in $\mathbf{e}$, will correspond to a *minimal*, but not necessarily minimum, contingency set $\mathbf{Y}$ for $\mathbf{x}$ in $\mathbf{e}$ (c.f. Section 4).

The generation of contingency sets is now useful for the computation of the inverse of the x-Resp score. For this we can use the built-in set-cardinality operation `#card(S,M)` of *DLV-Complex*. Here, `M` is the cardinality of set `S`. The score will be the result of adding 1 to the cardinality `M` of a contingency set `S`:

```
% computing the inverse of x-Resp
  invResp(E,U,R) :- cont(E,U,S), #card(S,M), R = M+1, #int(R).
```

For each counterfactual version of **e**, as represented by a model of the program, we will obtain a *local* x-Resp score. So, a particular feature value, `U`, may have several local x-Resp scores in different models of the program. For example, in the model corresponding to the change of humidity (and nothing else) we will get the atom `invResp(e,humidity,1)`. Finally, full explanations will be of the form `fullExpl(E,U,R,S)`, where `U` is a feature name, `R` is its inverse x-Resp score, and `S` is its contingency set (of feature names).

```
% full explanations:
  fullExpl(E,U,R,S) :-  expl(E,U,X), cont(E,U,S), invResp(E,U,R).
```

Following with our ongoing example, we will get in one model the atom `fullExpl(e,humidity,1,{})`. Additional information, such as the new feature values that lead to the change of label can be read-off from the associated model (examples follow). The original feature values can be recovered via the eid `e` from the original entity.

If we run the program starting with the original entity, we obtain ten different counterfactual versions of **e**. They are represented by the ten essentially different stable models of the program, and can be read-off from the atoms with the annotation `s`, namely: (with value changes underlined)

1. `ent(e,rain,high,high,weak,s)`
2. `ent(e,rain,high,high,strong,s)`, `ent(e,sunny,high,normal,strong,s)`, `ent(e,sunny,high,high,weak,s)`
3. `ent(e,rain,medium,high,strong,s)`, `ent(e,rain,low,high,strong,s)`, `ent(e,sunny,low,high,weak,s)`, `ent(e,sunny,medium,high,weak,s)`;
4. `ent(e,sunny,medium,high,strong,s)`, `ent(e,sunny,low,high,strong,s)`.

Below we show only three of the obtained models (the others are found in [7, Appendix B]). In the models we show only the most relevant atoms, omitting initial facts, intermediate probabilities, and chosen-related atoms:

```
M1 {ent(e,rain,high,normal,weak,o), ent(e,rain,high,normal,weak,tr),
    cls(e,rain,high,normal,weak,yes), ent(e,rain,high,high,weak,do),
    ent(e,rain,high,high,weak,tr), cls(e,rain,high,high,weak,no),
    ent(e,rain,high,high,weak,s), expl(e,humidity,normal),
    cont(e,humidity,{}),invResp(e,humidity,1),fullExpl(e,humidity,1,{})}
M2 {ent(e,rain,high,normal,weak,o), ent(e,rain,high,high,strong,tr),
    cls(e,rain,high,high,strong,no), ent(e,rain,high,high,strong,s),
    invResp(e,humidity,2), fullExpl(e,humidity,2,{wind}),
    invResp(e,wind,2), fullExpl(e,wind,2,{humidity})}
M3 {ent(e,rain,high,normal,weak,o), ent(e,sunny,high,normal,strong,tr),
    cls(e,sunny,high,normal,strong,no),ent(e,sunny,high,normal,strong,s),
    invResp(e,outlook,2), fullExpl(e,outlook,2,{wind}), ...}
```

The first model corresponds to our running example. The second model shows that the same change of the previous model accompanied by a change for Wind also leads to a change of label. We might prefer the first model. We will take care of this next. The third model shows a different combination of changes: for Outlook accompanied by Wind. In this model, the original Outlook value has $\frac{1}{2}$ as x-Resp score.

If we are interested only in those counterfactual entities that are obtained through a minimum number of changes, and then leading to maximum responsibility scores, we can impose *weak program constraints* on the program [23]. In contrast to hard constraints, as used above, they can be violated by a model of the program. However, only those models where the number of violations is a minimum are kept. In our case, the number of value differences between the original and final entity is minimized:

```
% weak constraints to minimize number of changes
  :~ ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), O != Op.
  :~ ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), T != Tp.
  :~ ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), H != Hp.
  :~ ent(E,O,T,H,W,o), ent(E,Op,Tp,Hp,Wp,s), W != Wp.
```

Running the program with them, leaves only model M1 above, corresponding to the counterfactual entity $\mathsf{e}' = \mathit{ent}(\mathsf{rain}, \mathsf{high}, \mathsf{high}, \mathsf{weak})$. This is a maximum-responsibility counterfactual explanation. □

6 Exploiting Domain Knowledge and Query Answering

CIPs allows for the inclusion of domain knowledge. In our example, describing a particular geographic region, it might be the case that there is never high temperature with a strong wind. Such a combination might not be allowed in counterfactuals, which could be done by imposing the program constraint:

```
:- ent(E,_,high,_,strong,tr).
```

If we run the program with this constraint, models M2 and M3 above would be discarded, so as any other where the inadmissible combination appears [8].

In another geographic region, it could be the case that there is a functional relationship between features, for example, between Temperature and Humidity: $\mathsf{high} \mapsto \mathsf{normal}$, $\{\mathsf{medium}, \mathsf{low}\} \mapsto \mathsf{high}$. In this case, from the head of the counterfactual rule, the disjunct ent(E,O,T,Hp,W,do) could be dropped for not representing an admissible counterfactual. Instead, we could add the extra rules:

```
ent(E,O,T,normal,W,tr) :- ent(E,O,high,H,W,tr).
ent(E,O,T,high,W,tr)   :- ent(E,O,medium,H,W,tr).
ent(E,O,T,high,W,tr)   :- ent(E,O,low,H,W,tr).
```

We can also exploit reasoning, which is enabled by query answering. Actually, the models of the program are implicitly queried, as databases (the models do not have to be returned, only the answers). Under the *cautious semantics* we obtain the answers that are true in *all* models, whereas under the *brave semantics*, the answers that are true in *some* model [23]. They can be used for different kinds of queries. The query semantics is specified when calling the program (naiveBayes.txt), so as the file containing the query (queries.txt):

```
\DLV>dlcomplex.exe -nofacts -nofdcheck -brave naiveBayes.txt queries.txt
```

If we do not use the weak constraints that minimize the responsibility, and we want the responsibility of feature Outlook, we can pose the query Q1 below under the brave semantics. The same to know if there is an explanation with less than 3 changes (Q2):

```
invResp(e,outlook,R)?                    %Q1
fullExpl(E,U,R,S), R<3?                  %Q2
```

Q1 returns 2, 3, and 4, then the responsibility for Outlook is $\frac{1}{2}$. Q2 returns all the full explanations with inverse score 1 or 2, e.g. e,outlook,2,{humidity}. We can also ask, under the brave semantics, if there is an intervened entity exhibiting the combination of sunny outlook with strong wind, and its label (Q3). Or perhaps, all the intervened entities that obtained label no (Q4):

```
cls(E,O,T,H,W,_), O = sunny, W = strong?       %Q3
cls(E,O,T,H,W,no)?                             %Q4
```

For Q3 we obtain, for example, e,sunny,low,normal,strong,yes; and for Q4, for example e,sunny,low,high,strong. We can ask, under the *cautions semantics*, whether the wind does not change under every counterfactual version:

```
ent(e,_,_,_,Wp,s), ent(e,_,_,_,W,o), W = Wp?       %Q5
```

We obtain the empty output, meaning Wind is indeed changed in at least one counterfactual version (i.e. stable model). In fact, the same query under the *brave semantics* returns the records where Wind remained unchanged, e.g. rain,high,high,weak, along with the original entity rain,high,normal,weak.

7 Final Remarks

Explainable data management and explainable AI (XAI) are effervescent areas of research. The relevance of explanations can only grow, as observed from- and due to the legislation and regulations that are being produced and enforced in relation to explainability, transparency and fairness of data management and AI/ML systems.

Still fundamental research is needed in relation to the notions of *explanation* and *interpretation.* An always present question is: *What is a good explanation?.* This is not a new question, and in AI (and other disciplines) it has been investigated. In particular in AI, areas such as *diagnosis* and *causality* have much to contribute. In relation to *explanations scores*, there is still a question to be answered: *What are the desired properties of an explanation score?*

Our work is about interacting with classifiers via answer-set programs. For our work it is crucial to be able to use an implementation of the ASP semantics. We have used *DLV*, with which we are more familiar. In principle, we could have used *Clingo* instead [20]. Those classifiers can be specified directly as a part of the program, as we did in our running example, or they can be invoked by a program as a external predicate [5]. From this point of view, our work *is not* about learning programs.

We have used in this paper a responsibility score that has a direct origin in *actual causality and responsibility.* When the features have many possible values, it makes sense to consider the proportions of value changes that lead to counterfactual versions of the entity at hand, and that of those that do not change the label. In this case, the responsibility score can be generalized to become an average or expected value of label differences [3, 5].

There are different approaches and methodologies in relation to explanations, with causality, counterfactuals and scores being prominent approaches that have a relevant role to play. Much research is still needed on the use of *contextual, semantic and domain knowledge*. Some approaches may be more appropriate in this direction, and we argue that declarative, logic-based specifications can be successfully exploited [5]. We have seen how easy becomes adding new knowledge, which would become complicated change of code under procedural approaches.

In this work we have used answer-set programming, in which we have accommodated probabilities as arguments of predicates. Probability computation is done through basic arithmetics provided by the *DLV* system. However, it would be more natural to explore the application of probabilistic extensions of logic programming [13, 29, 14, 22] and of ASP [1], while retaining the capability to do counterfactual analysis. In this regard, one has to take into account that the complexity of computing the x-Resp score is matched by the expressive and computational power of ASP [5].

Acknowledgments: Part of this work was funded by ANID - Millennium Science Initiative Program - Code ICN17002. Help from Jessica Zangari and Mario Alviano with information about *DLV* is very much appreciated.

References

[1] Baral, C., Gelfond, M. and Rushton, N. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 2009, 9(1):57-144.

[2] Bertossi, L. and Salimi, B. From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back. *Theory of Computing Systems*, 2017, 61(1):191-232.

[3] Bertossi, L., Li, J., Schleich, M., Suciu, D. and Vagena, Z. Causality-Based Explanation of Classification Outcomes. In *Proceedings of the Fourth Workshop on Data Management for End-To-End Machine Learning, DEEM@SIGMOD 2020*, pp. 6:1-6:10, 2020.

[4] Bertossi, L. An ASP-Based Approach to Counterfactual Explanations for Classification. In Proc. RuleML-RR 2020, Springer LNCS 12173, pp. 70-81.

[5] Bertossi, L. Declarative Approaches to Counterfactual Explanations for Classification. arXiv Paper 2011.07423, 2020. Journal submission after revisions.

[6] Bertossi, L. Score-Based Explanations in Data Management and Machine Learning: An Answer-Set Programming Approach to Counterfactual Analysis. Posted as Corr arXiv Paper 2106.10562. To appear in *Reasoning Web, 2021.*

[7] Bertossi, L. and Reyes, G. Answer-Set Programs for Reasoning about Counterfactual Interventions and Responsibility Scores for Classification. Extended version of this paper. arXiv Paper 2107.10159, 2021.

[8] Bertossi, L. and Geerts, F. Data Quality and Explainable AI. *ACM Journal of Data and Information Quality*, 2020, 12(2):1-9.

[9] Brewka, G., Eiter, T. and Truszczynski, M. Answer Set Programming at a Glance. *Commun. ACM*, 2011, 54(12):92-103.

[10] Calimeri, F., Cozza, S., Ianni, G. and Leone, N. Computable Functions in ASP: Theory and Implementation. Proc. ICLP 2008, Springer LNCS 5366, pp. 407-424.

[11] Calimeri, F., Cozza, S., Ianni, G. and Leone, N. An ASP System with Functions, Lists,and Sets. Proc. LPNMR 2009, Springer LNCS 5753, pp. 483-489.

[12] Chockler, H. and Halpern, J. Responsibility and Blame: A Structural-Model Approach. *Journal of Artificial Intelligence Research*, 2004, 22:93-115.

[13] De Raedt, . and Kimmig, A. Probabilistic (Logic) Programming Concepts. *Machine Learning*, 2015, 100(1):5-47.

[14] De Raedt, L., Kersting, K., Natarajan, S. and Poole, D. *Statistical Relational Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2016.

[15] Gelfond, M. and Lifschitz, V. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 1991, 9:365-385.

[16] Gelfond, M. and Kahl, Y. *Knowledge Representation and Reasoning, and the Design of Intelligent Agents*. Cambridge Univ. Press, 2014.

[17] Giannotti, F., Greco, S., Sacca, D. and Zaniolo, C. Programming with Non-Determinism in Deductive Databases. *Annals of Mathematics in Artificial Intelligence*, 1997, 19(1-2):97-125.

[18] Halpern, J. and Pearl, J. Causes and Explanations: A Structural-Model Approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 2005, 56(4):843-887.

[19] Halpern, J. Y. A Modification of the Halpern-Pearl Definition of Causality. In Proc. IJCAI 2015, pp. 3022-3033.

[20] Kaminski,, R., Romero, J., Schaub, T. and Wanko, P. How to Build Your Own ASP-based system?! arXiv:2008.06692, 2020.

[21] Karimi, A-H., von Kügelgen, B. J., Schölkopf, B. and Valera, I. Algorithmic Recourse under Imperfect Causal Knowledge: A Probabilistic Approach. In Proc. NeurIPS, 2020.

[22] Kimmig, A., Demoen, B., De Raedt, L., Santos Costa, V. and Rocha, R. On the Implementation of the Probabilistic Logic Programming Language ProbLog. *Theory and Practice of Logic Programming*, 2011, 11(2-3):235-262.

[23] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2006, 7(3):499-562.

[24] Lundberg, S., Erion, G., Chen, H., DeGrave, A., Prutkin, J., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. and Lee, S.-I. From Local Explanations to Global Understanding with Explainable AI for Trees. *Nature Machine Intelligence*, 2020, 2(1):2522-5839.

[25] Meliou, A., Gatterbauer, W., Moore, K. F. and Suciu, D. The Complexity of Causality and Responsibility for Query Answers and Non-Answers. Proc. VLDB 2010, pp. 34-41.

[26] Meliou, A., Gatterbauer, W., Halpern, J.Y., Koch, C., Moore, K. F. and Suciu, D. Causality in Databases. *IEEE Data Engineering Bulletin*, 2010, 33(3):59-67.

[27] Mitchell, T. M. *Machine Learning*. McGraw-Hill, 1997.

[28] Molnar, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book, 2020.

[29] Riguzzi, F. *Foundations of Probabilistic Logic Programming*. River Publ., 2018.

[30] Roth, A. E. (ed.) *The Shapley Value: Essays in Honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[31] Rudin, C. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, 2019, 1:206-215. Also arXiv:1811.10154,2018.

[32] Ustun, B., Spangher, A. and Liu, Y. Actionable Recourse in Linear Classification. In Proc. FAT 2019, pp. 10-19.