TU WIEN Informatics

Erklärungen für Klassifizierungen und Anfrageergebnisse mittels Maßzahlen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Florian Lackner, BSc
Matrikelnummer 11704916

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

Wien, 28. August 2023

Florian Lackner	Reinhard Pichler

TU WIEN Informatics

Score-based Explanations of Classification Outcomes and Answers to Database Queries

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Florian Lackner, BSc
Registration Number 11704916

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

Vienna, August 28, 2023

Florian Lackner	Reinhard Pichler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Florian Lackner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. August 2023

Florian Lackner

Danksagung

Ich möchte mich an dieser Stelle ganz herzlich bei meinem Professor, Reinhard Pichler bedanken, der mir während des gesamten Prozesses der Erstellung meiner Diplomarbeit mit Rat und Tat zur Seite stand. Seine fachliche Kompetenz und seine wertvollen Anregungen haben maßgeblich dazu beigetragen, dass ich mein Thema umfassend bearbeiten konnte.

Auch meiner Frau möchte ich danken, denn sie hat mich während der intensiven Schreibphasen unterstützt und mir immer wieder Mut gemacht, wenn ich an meine Grenzen gestoßen bin. Ohne ihre Geduld und ihr Verständnis wäre es mir nicht möglich gewesen, diese Herausforderung zu meistern.

Zusätzlich möchte ich mich bei Professor Leopoldo Bertossi bedanken, der durch mehrere Videotelefonate und seine Expertise maßgeblich zur Themenfindung beigetragen hat. Seine langjährige Erfahrung im Bereich Data Science und Artificial Intelligence hat mir wertvolle Einsichten vermittelt und mich inspiriert, meine Diplomarbeit auf viele seiner Arbeiten zu stützen.

Ich bin dankbar für all die Unterstützung durch Familie und Freunde. Nur durch sie war dieses Projekt umsetzbar.

Acknowledgements

I would like to take this opportunity to express my sincere thanks to my professor, Reinhard Pichler, who provided me with advice and support throughout the entire process of writing my thesis. His professional competence and his valuable suggestions have contributed significantly to the fact that I was able to comprehensively work on my topic.

I would also like to thank my wife, because she supported me during the intensive writing phases and always encouraged me when I reached my limits. Without her patience and understanding, it would not have been possible for me to overcome this challenge.

Additionally, I would like to thank Professor Leopoldo Bertossi, who contributed significantly to the topic identification through several video calls and his expertise. His many years of experience in the field of Data Science and Artificial Intelligence provided me with valuable insights and inspired me to base my thesis on much of his work.

I am grateful for all the support from family and friends. Only through them this project was feasible.

Kurzfassung

Maschinelles Lernen hat viele Aspekte moderner digitalisierter Unternehmen revolutioniert, aber die mangelnde Transparenz und Interpretierbarkeit dieser Modelle stellt eine Herausforderung dar. Wenn Entscheidungen auf der Grundlage von Modellen des maschinellen Lernens getroffen werden, sind Erklärungen wichtig, um die Gründe für die Ergebnisse zu verstehen. Diese Arbeit konzentriert sich auf Maßzahlen, die den Beitrag jedes Merkmals einer Entität zu ihrem Klassifikationsergebnis oder jedes Tupels in einer Datenbankreparatur messen, und untersucht insbesondere lokale Erklärungen, die Änderungen identifizieren, die erforderlich sind, um ein gewünschtes Ergebnis zu erzielen. Ziel ist es, die Undurchsichtigkeit von maschinell erlernten Modellen und Datenbankreparaturen zu beheben, indem Erklärungen auf der Grundlage genau definierter Werte erstellt werden.

Der SHAP-Score, der sich von den Shapley-Werten in der Spieltheorie ableitet, ist ein gut erforschter Score, der den Einfluss von Merkmalen auf das Ergebnis einer gegebenen Entität quantifiziert. Seine Anwendbarkeit auf bool'sche Schaltkreise und seine Berechnungskomplexität erfordern jedoch weitere Untersuchungen. Ein weiterer Score, der RESP-Score, führt das Konzept der Resposibility und der Kontingenzmengen ein, um die Verantwortung einzelner Merkmale zu bewerten. Die Komplexität dieses Scores ist in der Regel schwieriger als die des SHAP-Scores, jedoch deuten empirische Belege darauf hin, dass die Berechnung deutlich weniger Zeit in Anspruch nehmen kann. Um dies jedoch zu belegen, fehlen eine Implementierung und ein experimenteller Vergleich. Zudem konzentrieren sich diese Scores ausschließlich auf einzelne Merkmale, während der kombinierte Einfluss mehrerer Merkmale auf das Ergebnis ebenfalls sehr aussagekräftig ist.

Ziel dieser Arbeit ist es, die Definitionen der erklärenden Scores zu vereinheitlichen und ihre Berechnungskomplexität zu untersuchen. Darüber hinaus werden die Scores für Gruppen von Merkmalen erweitert. Um diese Ziele zu erreichen, wurde ein Python-Framework zur einfachen und intuitiven Berechnung der erklärenden Scores entwickelt. Die Beziehungen zwischen den Scores wurden analysiert und mögliche Erweiterungen und zukünftige Forschungsrichtungen diskutiert.

Abstract

Machine learning has revolutionized numerous aspects of modern digitalized businesses, but the lack of transparency and interpretability of these models poses challenges. When decisions are made based on machine learning models, explanations are crucial to understand the reasoning behind the outcomes. This study focuses on numerical scores that measure the contribution of each feature of an entity for its classification outcome or of each tuple in a database-repair, specifically exploring local explanations that identify changes required to achieve a desired outcome. The goal is to address the opacity of machine-learned models and database-repairs, by providing explanations based on well-defined scores.

The SHAP score, derived from Shapley values in game theory, is an extensively studied score that quantifies the influence of features on a specific entity's outcome. However, its applicability to boolean circuits and its computational complexity require further investigation. Another score, the RESP score, introduces the concept of responsibility and contingency sets to assess the responsibility of individual features. The complexity of this score is typically greater than that of the SHAP score, but empirical evidence suggests that the computation time can be significantly reduced. However, in order to demonstrate this, an implementation and an experimental comparison are currently absent. Moreover, it should be noted that these scores primarily emphasize individual features, whereas the combined impact of multiple features on the outcome is also highly significant.

This thesis aims to unify the definitions of explanation scores and investigate their computational complexity. Furthermore, the thesis extends the scores to handle sets of features. To achieve these goals, a Python framework was developed to compute the explanation scores in a simple and intuitive manner. Relationships between the scores were analyzed, and potential extensions and future research directions were discussed.

Contents

CHAPTER 1

Introduction

Machine learning has become an essential part of any digitalized business today. As a result, it influences our daily lives in many different ways. But often the models used are very intransparent and results are difficult to interpret.

Example 1.0.1. Consider a bank customer that wants to apply for a loan at her bank and gives all her data to check the liquidity. The bank employee rejects the customer because the learned model with the given data outputs a rejection.

Quite naturally, the question of "why?" arises. In order to be able to counter this, an answer is needed as to what changes the customer needs to make in order to get a loan after all. In other words an *explanation* is needed. These explanations can be quite different. In this work we want to focus on numerical scores, which measure how much of the result each feature of the machine learning model contributes. If this score is 0, a change in the data of this feature, would not entail any change for the result. On the other hand, if this score for a feature is set to 1, changing the data of that feature alone can achieve the desired result.

In the above example, the value "10" for the feature "age" would probably have a high score, as the customer being very young is very likely a reason for the rejection of the loan.

The desire is not to look at the totality of the data or to identify trends for the entire population. These explanations are called global explanations and are mainly used to explain the model to a developer or a manager. In this work, the focus is much more on the result of a single dataset. These local explanations tell which values in this single dataset have to be changed to get the desired result.

The problem is that the machine-learned model is often just a black box that cannot be fully understood. It is easy to calculate an output for an input, but it is not easy to find out how the model obtained the result. A very similar problem arises in the area of

databases: Given a database, a result to a query can be computed. But if someone asks for the important parts of the data that are crucial for the outcome, the answer is more difficult.

In practice, the mentioned scores can be useful when a database query produces an unexpected result. The individual scores of the data tuples can be used to understand whether this is a user mistake, an error in the data, or an inaccuracy in the query itself. This not only improves the user experience, but also helps identify potential problems with queries or data inconsistencies. Another practical application is in the area of network fault diagnosis. In a network of interconnected servers, individual servers may fail, resulting in a loss of connectivity. Using explanations, maintainers can effectively diagnose these failures. By testing subsets of servers and determining explanation scores, servers with high responsibility scores can be identified as potential culprits. This ranking allows maintainers to prioritize diagnostic work, leading to efficient troubleshooting and resolution of network problems.

In all of the highlighted examples, explanations provide valuable insight and transparency. These applications demonstrate the practical importance of explanations in artificial intelligence and database applications to improve decision making, problem solving, and system maintenance.

The SHAP score, which has undergone extensive research, has demonstrated its utility as a valuable tool in tackling the aforementioned problem areas. It is based on the game-theoretical model of Shapley values, which were defined by Lloyd Shapley as early as 1951. The Shapley value uses a wealth function to assign a score to each player based on how much that player contributes to the win. In the field of machine learning, the SHAP score can thus make a statement about an entity, how much one feature influences the result for this entity. This scheme can also be found in the field of databases, where the score determines how much influence certain data has on the final result of a query. For several classes of boolean circuits, in particular, for decision trees, the SHAP score can be computed in polynomial time [3, 4, 15] and has other useful properties such as additivity or efficiency [5]. For other boolean circuits it has been investigated in [15] and [3] that the computation of SHAP is #P-hard.

Quite differently, Bertossi et al. created a score starting from the definition of responsibility [13, 22, 24, 44]. It tries to quantify the responsibility of an individual feature for the outcome. First the COUNTER score and then the more mature RESP score was defined. [9] These do not consider only a single or a group of features, like the SHAP score, but work with the definition of causes and define contingency sets. [38] A contingency set specifies all that is required to ultimately determine the responsibility of a single feature. While there is a relationship between COUNTER and SHAP score, this is not yet known between RESP and SHAP score. [9] Furthermore, the very useful properties of the SHAP score, such as additivity and efficiency, have not yet been thoroughly investigated for COUNTER and RESP scores. However, there are already approaches to limit the complexity of the computation of the RESP score: in [9] a SQL-query was presented, which can compute the RESP score efficiently with fixed size of the contingency set.

In this thesis, our main aim is to unify the definitions of the mentioned explanation scores. Furthermore, we want to explore the limits of the computational complexity of these scores. This will lead us to extend the scores, such that they can also handle sets of features. In addition to Bertossi's implementation, we will implement our own Python framework to compute all explained scores in an straightforward and self-explanatory way. We will also discuss the relationships between the scores and their implementations, and think about extensions and future work.

One of the key challenges of the thesis was that there is significant variation in the definition of the scores in the literature. Consequently, we sought to establish a standardized and clearer description of these scores. Finally, by developing a common understanding of key terms and concepts, we were able to effectively compare the scores. In addition, we introduced two new scores by expanding the existing scores for sets of features. To facilitate a more comprehensive analysis of the scores, we implemented them, creating a new Python framework[1]. The implementation process involved carefully selecting appropriate data, training classifier models, and incorporating score functions that produce reasonable and intelligible results. By evaluating the implemented scores and examining the feature rankings, we gained valuable insight into the challenges associated with computing scores for larger instances. This, in turn, helped us refine the framework and prepare it for future research.

In Chapter 2, we explain the basics of machine learning and relational databases. After that, Chapter 3 is dedicated to a detailed consideration of explanations. In Chapter 4, we will look at causality from a philosophical perspective. In Chapters 5 and 6, scores and their computational complexity are examined in more detail, followed by a comparison and analysis of relationships between scores in Chapter 7. The definition of our newly designed scores is presented in Chapter 8. The implementation of all scores is explained in Chapter 9, while Chapters 10 summarizes all the findings and draws a conclusion. Finally, Chapter 11 is devoted to examining future research directions and articulating our perspectives on the potential avenues for achieving effective explanations.

[1] https://github.com/Florian-Lackner/score-based-explanations

CHAPTER 2

Preliminaries

In this chapter, we will start by defining terms and introducing research areas, which should help to better understand the following chapters. Furthermore, we want to point out details that will become important later or help to understand more complex topics.

2.1 Machine Learning

Nowadays, machine learning is used in many areas to solve complicated problems using given data. Algorithms try to learn how the solutions of certain problems look like in order to imitate them and to find solutions for similar problems. Mostly the algorithms get the problems as input and try to predict a solution. At the beginning of the learning phase this prediction is still bad, but over the training time, the algorithm ”learns” and improves. The use cases can be divided into three large areas:

- *supervised* learning: The algorithm receives inputs, each of which is labelled with a result. Through the results, the algorithm can check itself whether its own prediction was right or wrong and adapt accordingly for new similar inputs. [37]
- *unsupervised* learning: There is no labelled data. The algorithm has specifications and uses them to try to find out what a ”good” result looks like. [37]
- *reinforcement* learning: The algorithm is defined as an agent in a predefined world. Initially, it acts with the world aimlessly and arbitrarily, but after each action it receives feedback from the world through its sensors and learns through punishments and reinforcements how it should best behave in order to achieve a concrete goal in the world. [37]

Furthermore, the problems are divided into two groups. On the one hand, there is *classification*. As the name suggests, a so-called *classifier* classifies the inputs into

different groups. On the other hand, there are *regression* problems. These do not have a class as an output, but a continuous value. [35, 37]

Example 2.1.1. The best way to categorize bank customers according to their creditworthiness is to use a classifier. First, the classifier is given a lot of data about bank customers who have already received a loan and for whom it is known whether it was useful or not. The algorithm learns from the training data when a customer has been correctly or incorrectly categorized, and can thus adjust its future assumptions. After the long training period, the algorithm does not need to be adjusted and is able to correctly predict the category of new bank customers even without supervision.

Example 2.1.2. However, if the task is to determine how creditworthy a bank customer is, the solution is likely to be a percentage estimate rather than a category. A regression model works in a similar way to the classifier with training data. However, instead of adjusting the category, it adjusts internal values that lead to a higher or lower result in future predictions.

In general, the explanations mostly deal with supervised learning algorithms for classification results, so in this thesis we will mainly focus on this form of machine learning. However, it should be noted that there are other algorithms besides classification. For example, *logistic regression* and *neural networks* are widely used. These are rarely used for classification problems and are sometimes more complicated to compute, making them particularly difficult to explain. [35, 37]

2.1.1 Classification

Classifying entities can be done in many different ways. Entities $e \in \mathcal{E}$ are always the input for the classifier $\mathcal{C}$ and a label $\mathcal{C}(e)$, an associated class of e, is the output. Thus, a prediction is made as to which class the entity e belongs to. The features of an entity are taken into account. Each feature $f \in \mathcal{F}$ is a function which is applied to an entity e and assigns a value from its finite domain $dom(f)$. Thus $e = \langle f_1(e), f_2(e), \ldots, f_n(e) \rangle$ is a tuple of n values, which includes a value for each feature. [8]

A classifier $\mathcal{C}$ that returns one of only two possible labels for an entity e is called a *binary classifier*, so $\mathcal{C}(e) \in \{0, 1\}$ holds. In the following we will often deal with binary classifiers, since this is easier to understand, but many results can also be extended to arbitrary classifiers. Also the classifier in Example 1.0.1 was binary: the client can either get the credit or not. [8, 37]

An example for a classifier is a *decision tree*. It creates a tree from the training data, which only needs to be processed during prediction. Each node in the tree represents a query to the entity. The edges represent disjunctive answers for the queries. In order to be able to make a prediction, one starts with the root of the tree and selects the edge, which corresponds to the entity. In the reached leaf is then the class into which the instance was classified. In Section 3.2 in Figure 3.2 a decision tree is shown. [35, 37]

2.2 Relational Databases

A relational database is a data store with a *relational schema* $\mathcal{R}$. This consists of a domain of constants $\mathcal{C}$ and a finite set of predicates $\mathcal{P}$. Furthermore, $\mathcal{R}$ creates a first-order predicate logic language $\mathcal{L}(\mathcal{R})$ with an equality $=$. Besides constants, usually denoted by $a, b, c, \ldots$, there are also variables $x, y, z, \ldots$ and finite sequences of constants or variables $\vec{a}, \vec{b}, \vec{c}, \ldots, \vec{x}, \vec{y}, \vec{z}, \ldots$. $P \in \mathcal{P}$ consists of n-ary *atoms* of the form $P(t_1, t_2, \ldots, t_n)$, where the terms $t_1, t_2, \ldots, t_n$ can be both constants and variables. The term *ground* is used to describe atoms that do not contain variables. A finite set of ground atoms is then called *database* D for $\mathcal{R}$. [6]

2.2.1 Queries

Queries are first-order formulas that retrieve data from a database. A *Conjunctive Query (CQ)* Q is a query of the form $Q(\vec{x}) = \exists \vec{y}(P_1(\vec{x_1}) \wedge \cdots \wedge P_n(\vec{x_n}))$, where $P_i \in \mathcal{P}$ and $\vec{x} = (\bigcup \vec{x_i}) \setminus \vec{y}$ are different free variables. We denote a query Q with n free variables, which are replaced component-wise by $\vec{c}$, by $Q[\vec{c}]$. Such a sequence of constants, $\vec{c} \in \mathcal{C}^n$, is a so-called *answer* to Q from a database D, if $Q[\vec{c}]$ is `true` in D. The latter can also be written formally as $D \models Q[\vec{c}]$. The set of all answers to Q from D is $Q(D)$. If $Q(D)$ is not empty, then the query Q in the database D is said to *hold*. Furthermore, a *Boolean Conjunctive Query (BCQ)*, Q is a query without free variables with $Q(D) = \texttt{true}\}$ exactly if `true` is in D. In addition, conjunctive queries can also be written as *answer-set program (ASP)*, which we explain in more detail in Section 2.5. [8]

2.2.2 Database Repairs

Many databases hold data from a particular domain, often accompanied by *integrity constraints*, which in return should be satisfied by the database itself. These are usually BCQs that prohibit certain pairs of tuples and are therefore called denial constraints, DCs. If a database D does not follow its DCs, it is called *inconsistent* and must be repaired by a so-called *database-repair*. These repaired databases D' must have the same schema as database D, must satisfy all DCs so that $D' \models DCs$, and must differ minimally from D. [8]

Example 2.2.1. Consider a database D and a set of *denial constraints* DCs, both seen in Table 2.1. Because D does not satisfy the DCs, D is inconsistent. [8]

R	A
	a
	b

S	A	B
	a	c

T	A	C
	a	d

$$DCs : \neg\exists x \exists y(R(x) \wedge S(x, y))$$
$$\neg\exists x \exists y(R(x) \wedge T(x, y))$$

Table 2.1: An inconsistent Database. [8]

To repair the database we have to delete tuples from the database D to prevent the two joins in the DCs. Two other options would be to add or to modify tuples in D. If we would add new tuples, the original tuples, which are responsible for the inconsistent database, would continue to exist. Modifying tuples would be a way, but we do not deal with that in this thesis, because it would lead too far. [8]

Deleting tuples can be done in two ways: On the one hand, a consistent database can be built from tuples of the inconsistent database D until the database would be made inconsistent by each additional tuple. On the other hand, one could use D and delete as few tuples as possible from it to create a consistent database. It is important to avoid deleting additional tuples that do not need to be deleted, in other words, to keep as many tuples as possible from D so that the resulting database is still consistent. [8]

By both methods a so-called *S-repairs* is obtained. These are subset-maximal consistent subinstances of D. Examples of such repaired databases for Example 2.2.1 are: $D_1 = \{R(b), S(a, c), T(a, d)\}$ and $D_2 = \{R(a), R(b)\}$. Any tuple from D that could be added to these databases would make them inconsistent. Also note that D_1 and D_2 are the only possible S-repairs, since each tuple from D occurs in at least one repair. [8]

Another repair which should be mentioned here is the *C-repair*. These are maximum-cardinality consistent subinstances of D. So intuitively this is the largest subset of D that is still consistent. The only C-repair for Example 2.2.1 is D_1. Note that also with C-repairs, it is not possible to add more tuples from the database without making the repair inconsistent. If such addition were possible, there would exist a repair requiring fewer tuples to be removed from the database, indicating that the original repair was not a C-repair. Therefore, every C-repair is also an S-repair. However, as observed in D_2, the opposite does not hold. [8]

2.3 Probability Distributions

In the course of this work we will need some population distributions. These are important for making assumptions about the given data, as they are often incomplete or very large. Furthermore, the importance of an entity or tuple depends strongly on its frequency in the data. Entities or tuples that occur very often are more important than those that occur less often. The choice of algorithm also depends strongly on the data. If an input is known to be common, the algorithm can be designed to work well with that data. [15, 7]

In the following, we will discuss three distributions commonly found in the literature, along with their advantages and disadvantages. We will also discuss their explanatory relevance, as this is crucial for their use in this thesis.

2.3.1 Uniform Distribution

Probably the most natural choice of distribution is the *uniform distribution*. In this distribution, every tuple in the population $\mathcal{E}$ gets the same probability of $\frac{1}{|\mathcal{E}|}$. Since it

does not take into account that combinations of feature values may be more common than others, it can be argued that this is a rather poor choice for explanations. [7]

2.3.2 Fully-Factorized Distribution

Probably the most commonly used distribution for explanations is the *fully-factorized distribution*. As we will see in Chapter 5, it has good properties for calculating the SHAP score explained there.

Definition 2.3.1 (Fully-Factorized Distribution) [15, 7]. Let Pr be a fully-factorized distribution over n discrete and independent random variables $X_1, \ldots, X_n$ and for every instance $e = (f_1(e), \ldots, f_n(e)) \in \mathcal{E}$ we have

$$Pr(X_1 = f_1(e), \ldots, X_n = f_n(e)) = \prod_{i=1}^{n} Pr(X_i = f_i(e)).$$

where $Pr(X_i = f_i(e)) = \frac{|\{e' \in \mathcal{E} | f_i(e') = f_i(e)\}|}{|\mathcal{E}|}$. If $\mathcal{E}$ is too large or not sufficiently defined, a representative subset $S \subseteq \mathcal{E}$ can also be chosen to determine the probability of a value for a feature: $Pr(X_i = f_i(e)) = \frac{|\{e' \in S | f_i(e') = f_i(e)\}|}{|S|}$.

The fully-factorized distribution assumes that all features are independent and thus gains simplicity. This assumption is justified because classifiers often do not have enough information about the data and it can be assumed that all features are treated independently. Moreover, it has been shown several times that the full-factorized distribution is highly tractable and therefore has good preconditions. [15]

On the other hand, this distribution only takes the real population into account to a limited extent. Combinations of feature values, which occur more frequently together, are not included, since each feature is considered in isolation. [7]

2.3.3 Empirical Distribution

Another distribution discussed in the literature is the *empirical distribution*. Here, special attention is paid to the frequency of tuples in the empirically collected data. For this purpose, a sample $S \subseteq \mathcal{E}$ is chosen from the entire population $\mathcal{E}$, which is as representative as possible for the distribution of the entire population. The probability of a tuple in the distribution is set equal to the distribution in the sample S. If a tuple does not occur in sample S, its probability is 0. [7]

Definition 2.3.2 (Empirical Distribution) [7]. Let Pr be a empirical distribution over n discrete and independent random variables $X_1, \ldots, X_n$ and for every instance $e = (f_1(e), \ldots, f_n(e)) \in \mathcal{E}$ we have

$$Pr(X_1 = f_1(e), \ldots, X_n = f_n(e)) = \frac{|\{e' \in \mathcal{E} \mid f_1(e') = f_1(e), \ldots, f_n(e') = f_n(e)|\}}{|\mathcal{E}|}.$$

A major advantage of the empirical distribution is its low overhead. Due to the relatively small sample, there are many tuples with probability 0 that do not need to be considered further. On the other hand, this can also be a disadvantage, as we will see in Chapter 9. Unfortunately, it has been proved in [15] that computing the SHAP score with an empirical distribution is #P-hard and therefore intractable. [15]

2.4 Complexity classes

In computer science, it is not only important that there are algorithms to solve problems, but also that they are classified according to their efficiency. The classes into which we divide algorithms are called *complexity classes* and describe how complex a problem is. Besides problems that can be solved in polynomial time, which are in class P and are also called *tractable*, there are other complexity classes. In this thesis, our primary focus will be on the following complexity classes:

P: Problems which are solvable by a deterministic Turing machine in polynomial time. [18]

NP: Problems which are solvable by a non-deterministic Turing machine in polynomial time. [18]

#P: Problems which count how many solutions there are to a problem solvable by a non-deterministic Turing machine in polynomial time. [42]

2.4.1 Polynomial Hierarchy

An oracle is an extension of a Turing machine. A Turing machine with an oracle for the complexity class C can solve problems of C in only one computation step. The so-called *polynomial hierarchy* arranges different complexity classes in a hierarchy with so-called *levels*. The levels $\sum_0$P, $\sum_1$P, $\sum_2$P, etc. are defined inductively in such a way that problems of level $\sum_{i+1}$P can be solved by a deterministic polynomial-time Turing machine with an oracle for a problem of level $\sum_i$P. The complexity classes described above can also be placed in this hierarchy as $\sum_0$P = P and $\sum_1$P = NP. [41]

Furthermore, it is relevant how often this oracle is used. So we define the following notation:

Definition 2.4.1 [36]. Let C be a complexity class, then the complexity class $FP^{C[log\ n]}$ consists of all functions that can be computed by a polynomial-time Turing machine with an oracle for a problem in C, asking a total of $O(log|x|)$ queries for the input x.

2.4.2 Polynomial Time Reduction

In order not to start from scratch when estimating the complexity of a problem, researchers like to use so-called *reductions*. Reduction is a method of theoretical computer science

in which one problem is reduced to another. If there is an algorithm for the second problem, the first one can also be solved by reduction. The reducibility is therefore a relation on the set of problems, by which the complexity of two problems can be related to each other. A polynomial-time reduction from a problem A to a problem B, denoted by $A \leq^P B$, is a polynomial-time algorithm that solves problem A with access to an oracle for problem B. This kind of reductions is also called *Cook reduction*. Furthermore, $A \equiv^P B$ holds if $A \leq^P B$ and $B \leq^P A$. [15, 18]

2.4.3 Completeness and Hardness

The concept of reductions serves as the basis for determining the difficulty of a problem within a complexity class. A problem X is considered *hard* for a complexity class C if every problem in C can be reduced to X in polynomial time. Consequently, no problem in C can be more difficult than X, as an algorithm for X enables us to solve any problem in C with only a polynomial slowdown. Notably, the collection of problems that are hard for the complexity class NP is referred to as the set of NP-hard problems. [18]

If a problem X is both hard for a complexity class C and is a member of C, it is referred to as being *complete* for C. In this context, X represents the most complex problem within C, as it is at least as hard as the hardest problems in C. In particular, the class of NP-complete problems is important because it contains the hardest problems in the complexity class NP. Since every problem in NP can be reduced to an NP-complete problem in polynomial time, the discovery of an NP-complete problem that can be solved in polynomial time would imply the equivalence of complexity classes P and NP. [18]

Example 2.4.2. Consider again the well-known NP-complete problem SAT. The questions of the associated problems of the mentioned complexity classes are:

P: For a given propositional formula, is Φ a satisfying variable assignment?

NP: Does a satisfying variable assignment exist for a given propositional formula?

#P: How many satisfying variable assignments exist for a given propositional formula?

2.5 Answer-Set Programming

ASPs Π are finite sets of *rules* of the form:

$$A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_l$$

where $n, m, l \geq 0$ and A_i, B_j, C_k are *atoms*. These Atoms are of the form $P(\vec{v})$, where P is a predicate of fixed arity and $\vec{v}$ a vector of variables and constants of the same arity. $A_1 \vee \cdots \vee A_n$ is called the *head* and $B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_l$ is called the *body* of the rule, where $B_1, \ldots, B_m$ are positive literals and $not\ C_1, \ldots, not\ C_l$ are negative literals. All variables appearing in $A_1, \ldots, A_n, not\ C_1, \ldots, not\ C_l$ also have to be in $B_1, \ldots, B_m$. [8]

All constants of Π form the *Herbrand universe* $H(\Pi)$. To obtain a *ground rule* from a rule in Π, each variable has to be instantiated with all possible values in $H(\Pi)$. The *ground* version of Π $gr(\Pi)$ is an ASP with only ground rules. Furthermore, the *Herbrand base* $HB(\Pi)$ consists of all these instantiated atoms with constants. A *model* M of Π is a subset of $HB(\Pi)$ which satisfies $gr(\Pi)$ that means for every ground rule $A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_l$ of $gr(\Pi)$, if $\{B_1, \ldots, B_m\} \subseteq M$ and $\{C_1, \ldots, C_l\} \cap M = \emptyset$, then $\{A_1, \ldots, A_n\} \cap M \neq \emptyset$. Besides, M is a *minimal model* when there is no other model M' with $M' \subset M$. The set of minimal models of Π is denoted by $MM(\Pi)$. Now for a subset $S \subseteq HB(\Pi)$, the Gelfond-Lifschitz reduct can be constructed as described in [19]. This involves deleting any rule from $gr(\Pi)$ that contains negative occurrences of atoms from S, as well as removing negative atoms from all remaining rules. If $S \in MM(GL_S(gr(\Pi)))$, it is referred to as a *stable model* or *answer-set*. Furthermore, every stable model is also a minimal model. [19]

There are *stratified* and *unstratified* ASPs, which differ in their complexity. Unstratified ASPs have at least one cycle in the definition of predicates, which contains negation. Thus, these programs cannot be evaluated iteratively, since first it must be checked whether such a cycle exists and whether the individual atoms may exist at all. This is different by stratified programs. Here one starts with the *facts*, rules without bodies, where the arrow for generation can be omitted. By applying the remaining rules, new atoms can then be created step by step. These steps are often represented as layers and are also called *strata*. [8]

Example 2.5.1. The following program is unstratified, because a,b and c form a cycle with negation:

$$a \leftarrow not\ b; b \leftarrow c; c \leftarrow a$$

Furthermore, the evaluation of an ASP is divided into two forms. On the one hand there is the *brave semantics*, which includes all atoms, which are contained in at least one model. On the other hand there is the *skeptical (or cautious) semantics*, which includes only the ground atoms which are contained in all models. [8]

An example shows the difference between those:

Example 2.5.2. Consider the following ASP Π:

$$a \vee b \leftarrow c, not\ e; c \leftarrow d; d$$

The two stable models of P are $\{a, c, d\}$ and $\{b, c, d\}$, where each set includes only the atoms which are `true` and exclude all atoms which are `false`. The brave semantics would then output $\{a, b, c, d\}$ and the skeptical semantics $\{c, d\}$.

CHAPTER 3

Explanations

Our digital systems have become very complex these days. So complex, in fact, that we are often no longer able to calculate exactly what is mathematically happening in the process. In this chapter, we want to introduce two such problems. Both are about a system that gets an input and returns an output. If the output has to be changed to a specified value, one has to know which input is necessary for it. To find a clever way to use the right input is a big problem in computer science in our time.

To explain, in the end, to a person why a certain part of the input was decisive for the output, this must be calculated. The results are called *explanations*. Explanations are more and more often given by *score-based methods*. These assign a *score* to each part of the input to show which part contributed how much to the output. If the value is very high, this part is essential for the outcome. If the value is very low, or even zero, it might even be omitted and the result would remain the same. [7, 8]

These score-based methods for explanations often use so-called *counterfactual interventions* to get their scores for each part of the input. Counterfactual interventions are small changes in the initial state to get back a desired result. If the result really changes due to a counterfactual intervention, the changed part of the input is called *counterfactual causes*. If the input is changed only slightly, but the rest remains the same, it may still be that the result changes. If it changes to the desired result, it is known that this part of the input contributed to the original result. [8]

The information and the combination of these counterfactual interventions often have great influence for the scores to extract explanations for the given input. If there were a set of multiple values that would need to be changed to get the desired outcome it is called a *contingency set*. Causes that require a contingency set to specifically change the outcome are called *actual causes*. Note that an actual cause with an empty contingency set is a counterfactual cause. This makes every counterfactual cause an actual cause. A larger contingency set also means that each individual part of the input is less relevant and

thus has a smaller score. However, a lower score also means that the resulting explanation is worse, since the score quantifies the relevance for the result. Thus, following the types of database repairs, there are two types of explanations that we distinguish. On the one hand, there are *s-explanations* which contain only contingency sets whose proper subsets would not cause a change in the result. There may also be several different contingency sets of different sizes. On the other hand, there are *c-explanations* which minimise the size of the contingency sets and thus have higher scores. So every c-explanation is also an s-explanation and not necessarily unique. [7, 8]

3.1 Explanations for Classification

Consider a classifier $\mathcal{C}$ which reads in an entity with n features $\mathcal{F} = \langle f_1, f_2, \ldots, f_n \rangle$. Often, one does not know every detail about the classification of $\mathcal{C}$, especially the mathematical model inside, and most importantly, the data it was trained on. Thus such a classifier can be seen as a black-box. One only has access to the input and the output and can thus draw conclusions about how things work inside the classifier. In Figure 3.1 this scheme is shown with $\mathcal{C}$ as black-box classifier. [6, 8]

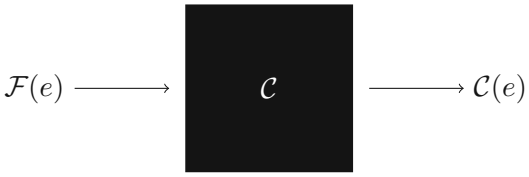

Figure 3.1: A black-box Classifier.

If an explanation is required and the classifier would be a decision tree or a logistic regression, it would perhaps make more sense to give the explanation on the basis of the internals of the model. But without a decision tree, but with a very complicated neural network, for example, it's very hard to make inferences, even if the internals of the classifier are known. To cover all cases, we assume the "worst case", where we do not know which model is used and still have to give an explanation. [8]

Example 3.1.1. Consider a person who always rides a bike to work. However, she does not ride when the weather is too extreme, that is, when it is too hot or when it is raining and the wind is strong.

Example 3.1.1 shows well what kind of explanations score-based methods provide. Given a weather situation, one either rides a bike or not. From an outsider's perspective, one does not know what weather condition would have to prevail for the bicycle to be used. A decision tree, such as the one shown in Figure 3.2, can help. The set of features is

$\mathcal{F} = \{outlook, humidity, wind\}$ and the possible values are

$$\begin{aligned} dom(outlook) &= \{sunny, cloudy, rain\}, \\ dom(humidity) &= \{high, normal\}, \\ dom(wind) &= \{strong, weak\}. \end{aligned}$$

Each weather condition is considered an entity and contains values for each of the three features. The decision tree now indicates with "yes" if it is a weather condition where it is comfortable to ride a bike or with "no" if it is not. For example, the label for weather condition $\{sunny, normal, weak\}$ is "yes". [8]

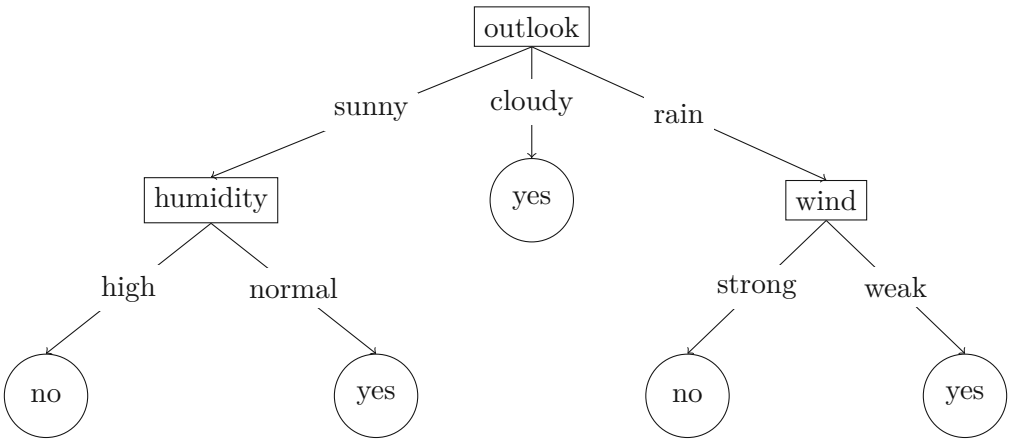

Figure 3.2: A Decision Tree for Example 3.1.1. [8]

In order to give an explanation, one only has to search the leaves of the tree for the desired result and form an entity from the path to the root. Without the information about the decision tree, however, this task is more difficult and counterfactual interventions must be used. By starting with the weather $e = \{rain, normal, strong\}$, we implement counterfactual interventions into our Example 3.1.1 to determine the responsibility of all parts of the weather. For the weather condition e the classifier outputs "no", so it is not comfortable to ride a bike in this weather condition. However, if we *hypothetically intervene* the value for outlook from "rain" to "sunny", we get a *counterfactual version* $e' = \{\underline{sunny}, normal, strong\}$ of e. Using the classification of the decision tree for e', it returns "yes". So we know that the outlook alone is a reason why the bike is not used, and thus we know that "rain" was very relevant for the original classification of e. In addition, the value for the wind could also be changed, as the desired result can then be achieved without altering the outlook. Reconciling all these issues is the task of the aforementioned explanatory scores. [6, 8]

3.2 Explanations in Databases

The same questions about classifiers in the machine learning domain are also raised in the field of databases. In the following section we will describe a similar problem for databases.

Example 3.2.1. Consider a database D, with relations $R : \{R(a,b), R(b,b), R(c,d)\}$ and $S : \{S(a), S(b), S(c)\}$ and a BCQs $Q : \exists x \exists y(S(x), S(y), R(x,y))$ as seen in Table 3.1, then $D \models Q$ holds. If we assign values from the database to the variables of Q and see if these relations exist in the database, we quickly find at least one cause for $D \models Q$: $\{S(a), S(b), R(a,b)\}$. [8]

R	A	B
	a	b
	b	b
	c	d

S	C
	a
	b
	c

$$Q : \exists x \exists y(S(x), S(y), R(x,y))$$

Table 3.1: A Database and a BCQ. [8]

Just as before, the question is how responsible are these tuples and which other tuples in the database are responsible for $D \models Q$ to hold? Here, *counterfactual causes* are tuples why a query is `true` or $false$. A tuple $\tau \in D$ is a counterfactual cause for a query Q if $D \models Q$ and $D \setminus \{\tau\} \not\models Q$ holds. In Example 3.2.1, $S(b)$ is a counterfactual cause for Q because Q would no longer be `true` if $S(b)$ is removed from the database. [8]

If removing a single tuple is not enough to change the result of a query, perhaps multiple tuples will do it. If there is a tuple $\tau \in D$ and a contingency set $\Gamma \subseteq D$ such that τ is a counterfactual cause for Q in $D \setminus \Gamma$, then τ is an *actual cause*. In Example 3.2.1, $R(a,b)$ is an actual cause for Q with the contingency set $\{R(b,b)\}$. Q would still be `true` if $R(a,b)$ or $R(b,b)$ were removed from the database, but not both. Furthermore, $R(a,b)$ becomes a counterfactual cause for Q if the contingency set, $\{R(b,b)\}$ is removed from the database. [8]

3.2.1 Database Repairs

Very closely related to counterfactual causes in databases are database repairs. The question that arises is: *What are the consistent answers of a query Q to a database D that does not satisfy a given set of denial constraints DCs?* The answer to this can be found in the following example, which shows that causes and database repairs have a lot of similarity:

Example 3.2.2. Consider a BCQ

$$Q : \exists \vec{x}(P_1(\vec{x}_1) \wedge \cdots \wedge P_m(\vec{x}_m)),$$

and a database D for which $D \models Q$ holds. Now it is possible to find out the reason for $D \models Q$ and the contingency set by database repairs.

By negating Q a denial constraint DC can be created:

$$DC : \neg\exists\vec{x}(P_1(\vec{x}_1) \wedge \cdots \wedge P_m(\vec{x}_m)).$$

Thus, if $D \models DC$ holds, then D is inconsistent. Now, to get the cause for $D \models Q$ with minimal contingency sets, one only needs to do an S-repair for D and DC. If there is a restriction to the cause for $D \models Q$ to find a cause with minimum contingency sets, and maximum responsibilities, then a C-repair for D and DC helps. This is because if a tuple is responsible for an inconsistent database, then deleting it is a way to make the database consistent again.

CHAPTER 4

Causality

We have dealt a lot with causes and responsibilities so far, but what these two terms really mean has not yet been clearly defined. Therefore, in this chapter we will deal with them and show that these definitions can only be expressed very vaguely. Since these two terms come from the intersection of computer science and philosophy, we will refer specifically to [24], [22] and [44], the results of the work of Judea Pearl and Joseph Halpern.

4.1 Causal Dependence

In 1973 Lewis described causality in [29] very intuitively and used the following notation: If two events happen in our actual world, we call them cause C and event E, where E depends on C, hence $C > E$ holds. So if C happens, then E also happens. Consequently, he arrived at a definition of causality:

Definition 4.1.1 (Causal Dependence) [44]. Event E is causally dependent on C if and only if $C > E$ and $\neg C > \neg E$ holds.

Example 4.1.2. An example of this is a lightning striking a forest and starting a devastating fire. The first event is the strike of the lightning, which causes a second event, the forest fire. If the lightning did not strike the forest, the fire would not occur. Therefore, the fire is causally dependent on the lightning. [29]

Lewis himself understood that this condition is only sufficient, but not necessary, because *overdetermination* shows that there must be more than that.

4.2 Overdetermination

Example 4.2.1. Consider a lightning striking a forest and an arsonist setting a fire in

the same forest at the same time by dropping a match. Both events together and each individually lead to the ensuing forest fire. [44]

Overdetermination means that there are several events leading to the same situation. In Example 4.2.1, the forest would still catch fire if there had been no lightning due to the arsonist's match. According to Lewis' definition, this would mean that the lightning strike would not have been a cause of the forest fire. Conversely, if the arsonist did not set the fire, the forest fire would also be caused by the lightning, so the match of the arsonist could not have been a cause either. Nevertheless, the lightning and the arsonist each are causes of the forest fire. Lewisian causal dependency is therefore not necessary for actual causality. [44]

4.3 Definition of Halpern and Pearl

Halpern and Pearl further developed Lewis' definition of causal dependence to become probably the most widely used definition of actual causes because it solves the problem of overdetermination. It was originally proposed in [24] and later updated in [22]. However, as we will see in the next section, this definition fails in another place, namely the problem of isomorphism. [44]

For their definition, Halpern and Pearl tried to better model the world in which events happen and first introduced causal models:

Definition 4.3.1 (Causal Model) [44]. A causal model is a 4-tuple $M = (\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{E})$, where

1. $\mathcal{U}$ is a set of exogenous variables;
2. $\mathcal{V}$ is a set of endogenous variables;
3. $\mathcal{R}$ is a set of functions assigning a set of possible values $\mathcal{R}(X)$ to each variable X in $\mathcal{U}$ or $\mathcal{V}$;
4. $\mathcal{E}$ is a set of structural equations, where $\mathcal{E}$ associates each endogenous variable $X \in \mathcal{V}$ with a function f_X, such that

$$f_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{V \in \mathcal{V} - \{X\}} \mathcal{R}(V)) \to \mathcal{R}(X)$$

Exogenous variables describe the context of the world. In Example 4.2.1 the context would include all factors that influence the actual events, but are not mentioned because the reader takes them for granted. For example, an exogenous variable could be the oxygen in the air, which is necessary for a forest fire, or the intention of the arsonist. [44]

Endogenous variables, on the other hand, represent all events that are the origins or consequences in a causal scenario. So in Example 4.2.1 these would be

$$\begin{aligned} L &: 1 \text{ if the lightning strikes, 0 if not} \\ M &: 1 \text{ if the arsonist drops a match, 0 if not} \\ FF &: 1 \text{ if there is a forest fire, 0 if not} \end{aligned}$$

Each of these events can either occur or not and therefore they get either the value 1 or 0 respectively. A *possible world* is an assignment of values to all variables in $\mathcal{V}$. So in Example 4.2.1 a possible world would be the *actual world* $\{L = 1, M = 1$ and $FF = 1\}$. [44]

Since it often happens that a cause consists of several events, from now on we will abbreviate conjunctions of multiple primitive events $X_1 = x_1 \wedge X_2 = x_2 \wedge \cdots \wedge X_n = x_n$ by $\vec{X} = \vec{x}$. Here $\vec{X}$ is the vector of variables $(X_1, X_2, \ldots, X_n)$ and $\vec{x}$ is the vector of values $(x_1, x_2, \ldots, x_n)$.

Furthermore, structural equations exist exclusively for endogenous variables. Thus, the exogenous variables are taken as given. The structural equation from Example 4.2.1 is $FF = L \vee M$. This expresses that both events, the lightning strike and the falling match, individually and together lead to the forest fire. Only if both events do not happen, the forest will not catch fire. Moreover, these structural equations are not cyclic by definition, otherwise an event could be its own cause. [44]

A *causal setting* $(M, \vec{u})$ is a model M with a value assignment for all exogenous variables $\mathcal{U}$. The relation $(M, \vec{u}) \models \varphi$, where M is a model and φ is any boolean combination of primitive events, is then called *satisfaction relation*. This satisfaction relation holds if and only if φ holds after the exogenous variables $\mathcal{U}$ are set to the values in $\vec{u}$. In Example 4.1.2, first the context is set in which the lightning can strike, then the assignment of the endogenous variables begins through the structural equations $\mathcal{E}$ in M.

Moreover, we define *interventionist counterfactuals* $\vec{X} = \vec{x}$ in $(M, \vec{u}) \models [\vec{X} = \vec{x}]\varphi$, which holds exactly if and only if $(M_{\vec{X}=\vec{x}}, \vec{u}) \models \varphi$ holds, where $M_{\vec{X}=\vec{x}}$ means M with additional structural equations $\vec{X} = \vec{x}$. To do this, the structural equations for the variables from $\vec{X}$ must be deleted beforehand and then $\vec{X} = \vec{x}$ must be added. [44]

For instance, in Example 4.1.2 this would be if we wanted to do a thought experiment by asking what would have happened if the lightning did not strike the forest. Formally:

$$\begin{aligned} (M, \vec{u}) &\models [L = 0](FF = 0) \\ (M_{L=0}, \vec{u}) &\models (FF = 0) \end{aligned}$$

These two satisfaction relations are equivalent because L can be set to a value either already in the model by a structural equation or only in the assignment of the endogenous variables to create a possible world. They also hold because the forest fire does not start without the lightning.

It is possible to transfer the Lewisian Causal Dependence into a causal model:

Definition 4.3.2 (Lewisian Causal Dependence) [44]. Event $E = e$ is causally dependent on cause $C = c$ in $(M, \vec{u})$, if and only if $(M, \vec{u}) \models [C = c](E = e)$ and there are c' and e', with $c' \neq c$ and $e' \neq e$ such that $(M, \vec{u}) \models [C = c'](E = e')$.

However, as already described, Lewisian Causal Dependence is not necessary for causality, so we are interested in the definition of Halpern and Pearl:

Definition 4.3.3 (Actual Cause) [24]. Let M be a causal model and let $\vec{u}$ be a value combination for the exogenous variables $\mathcal{U}$. Then $\vec{X} = \vec{x}$ is an actual cause of φ in the causal setting $(M, \vec{u})$ if the following three conditions hold:

AC1. $(M, \vec{u}) \models (\vec{X} = \vec{x})$ and $(M, \vec{u}) \models \varphi$

AC2. There exists a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}', \vec{w}') \neq (\vec{x}, \vec{w})$ of the variables in the subpartition $(\vec{X}, \vec{W})$ such that $(M, \vec{u}) \models (\vec{Z}' = \vec{z})$ and

- **(a)** $(M_{\vec{W}=\vec{w}'}, \vec{u}) \models [\vec{X} = \vec{x}']\neg\varphi$
- **(b)** $(M_{\vec{W}=\vec{w}'}, \vec{u}) \models [\vec{X} = \vec{x}, \vec{Z}' = \vec{z}]\varphi$ for all $\vec{Z}' \subseteq \vec{Z}$.

AC3. $\vec{X}$ is minimal; no proper subset of $\vec{X}$ satisfies conditions **AC1** and **AC2**

AC1 in words describes that both the cause and the consequences must have happened. Without one of the two, there is no cause or no effect on an event, so there can be no causal connection. **AC3** states that the cause is minimal. If a subset of the cause would satisfy condition **AC1** and **AC2**, then this alone is the cause and further events are not necessary to determine the causal relationship. However, **AC2** brings up a new idea. What was called a contingency set in Chapter 3 is now all events in $\vec{W} = \vec{w}$. In other words, **AC2(a)** defines that to prevent the consequence φ both $\vec{X}$ and $\vec{W}$ must be assigned different values $(\vec{x}', \vec{w}')$ than their original values $(\vec{x}, \vec{w})$. But in addition, as **AC2(b)** describes, $\vec{X} = \vec{x}$ is the actual cause for φ, even if $\vec{W}$ gets different values $\vec{w}'$ and no matter which subset $\vec{Z}' \subseteq \vec{Z}$ were assigned to their values of the actual world. [13, 44]

Example 4.3.4. Let us consider Example 4.2.1 with the definition of Halpern and Pearl. We recall the three endogenous variables $\mathcal{V} = \{L, M, FF\}$, which stood for the lightning strike, the dropping of the arsonist's match, and the forest fire. We want to check if the lightning strike $L = 1$ was an actual cause for the forest fire $FF = 1$. Thus, in our model **AC1** is `true` because both $(M, \vec{u}) \models (L = 1)$ and $(M, \vec{u}) \models (FF = 1)$ are `true`. To look at **AC2** we need a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$. For this we choose $\vec{W} = \{M\}$ as contingency set with $\vec{w}' = 0$, leaving $Z = \{L, FF\}$. By choosing this contingency set, **AC2(a)** is satisfied, since $(M_{M=0}, \vec{u}) \models [L = 0](FF \neq 1)$. No matter which subset $\vec{Z}' \subseteq \vec{Z}$ is taken, **AC2(b)**: $(M_{M=0}, \vec{u}) \models [L = 1, \vec{Z}' = \vec{z}](FF = 1)$ is always `true`. Because **AC1** and **AC2** are `false` for all proper subsets of the actual cause $L = 1$, **AC3** is `true`. Thus all conditions are satisfied, so the lightning strike is an actual cause of the forest fire. [44]

4.4 Problem of Isomorphism

The problem with Halpern and Pearl's definition is that by creating the causal model, information about the actual scenario is lost. As a result, two similar scenarios with the same model can never get different causal judgments. This is called the *problem of isomorphism*. An example of such an isomorphism is the *bogus prevention*, which will be discussed in the following section. [44]

4.4.1 Bogus Prevention

Example 4.4.1. Consider a pedestrian who wants to cross a street. Because he is distracted, he does not see an approaching car. Worried about the pedestrian, a witness pulls him back onto the sidewalk in time, thus saving his life. Fortunately, however, the car is able to stop quickly enough and would never have hit the pedestrian. Intuitively, the witness did not save the pedestrian from death because the witness was just too cautious and the car could in fact stop in time.

The causal model for Example 4.4.1 looks like this:

$$\begin{aligned} H &: 1 \text{ if the car hits the pedestrian, 0 if not} \\ W &: 1 \text{ if the witness pulls back the pedestrian, 0 if not} \\ PS &: 1 \text{ if the pedestrian survives, 0 if not} \\ \mathcal{E} &: PS = \neg H \vee W \\ \text{actual World} &: \{H = 0, W = 1, PS = 1\} \end{aligned}$$

Since the models of Example 4.2.1 and Example 4.4.1 can be transformed into each other, they are isomorphic. We can replace W, $\neg H$ and PS from Example 4.4.1 with L, M and FF from Example 4.2.1 respectively. Both the causal setting and the structural equations are identical and, using Halpern and Pearl's definition, lead to the same causal judgment. With the contingency set $\{H = 1\}$, $W = 1$ is a cause of $PS = 1$, just as $L = 1$ was a cause of $FF = 1$ in Example 4.2.1 when $\{M = 0\}$ was the contingency set. This is counterintuitive, however, because the witness simply misjudged the situation and the pedestrian would have survived even without his intervention. That the witness' *bogus prevention* was the cause of the pedestrian's survival is an indication that the Halpern and Pearl definition establishes causal relationships in some cases that do not hold. Therefore, it was necessary to extend the definition of the causal model to include this. [44]

4.4.2 Normality of Witness Worlds

Halpern and Hitchcock addressed the issue of bogus prevention in [23] by believing that for every causal inference there is a counterfactual world of contingency $\vec{W} = \vec{w}'$ which witnesses the actual causality. However, this counterfactual world should be "at least as normal as the actual world" [23]. They defined normality $\succeq$ as an order of all possible

worlds that can arise from $\mathcal{V}$ and $\mathcal{R}$ from Definition 4.3.1, and thus came up with the following definition. [44]

Definition 4.4.2 (Extended Causal Model) [23]. An extended causal model is a 5-tuple $EM = (\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{E}, \succeq)$, where

1. $\mathcal{U}$ is a set of exogenous variables;
2. $\mathcal{V}$ is a set of endogenous variables;
3. $\mathcal{R}$ is a set of functions assigning a set of possible values $\mathcal{R}(X)$ to each variable X in $\mathcal{U}$ or $\mathcal{V}$;
4. $\mathcal{E}$ is a set of structural equations, where $\mathcal{E}$ associates each endogenous variable $X \in \mathcal{V}$ with a function f_X, such that
$$f_X : (\times_{U \in \mathcal{U}} \mathcal{R}(U)) \times (\times_{V \in \mathcal{V} - \{X\}} \mathcal{R}(V)) \to \mathcal{R}(X)$$
5. $\succeq$ is a normality partial order on all possible worlds generated by $\mathcal{V}$ and $\mathcal{R}$.

Hitchcock and Knob noted in [25] that people's intuitions about causal inference are often influenced by statistical and moral norms. Thus, whether one world is more *normal* than another Halpern and Hitchcock described in [23] lies in their events. Events are more normal than others if they occur statistically more often or if they are more moral. For example, it is not very common for arsonists to set fire to a forest, as this behavior is morally incorrect and also does not occur very often. Therefore, we can assume that the world $\{L = 1, M = 0, FF = 1\}$ is more normal than the world $\{L = 1, M = 1, FF = 1\}$. Furthermore, there are events which are not comparable, resulting in incomparable worlds. Therefore, the normality order is a partial order and not a total order. [44]

We define s as the world which is characterized by the causal setting $(EM, \vec{u})$ and $s_{\vec{W}=\vec{w}'}$ as the world which is characterized by $(EM_{\vec{W}=\vec{w}'}, \vec{u})$. The difference between s and $s_{\vec{W}=\vec{w}'}$ is that the latter has the redefined structural equations $\vec{W} = \vec{w}'$ in its extended model EM. Thus, it is now possible to define actual causes also with extended causal models. [44]

Definition 4.4.3 (Actual Cause with Extended Causal Model) [23]. Let EM be an extended causal model and let $\vec{u}$ be a value combination for the exogenous variables $\mathcal{U}$. Then $\vec{X} = \vec{x}$ is an actual cause of φ in the causal setting $(EM, \vec{u})$ if the following three conditions hold:

AC1$^{\mathbf{EM}}$. $(EM, \vec{u}) \models (\vec{X} = \vec{x})$ and $(EM, \vec{u}) \models \varphi$

AC2$^{\mathbf{EM}}$. There exists a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}', \vec{w}') \neq (\vec{x}, \vec{w})$ of the variables in the subpartition $(\vec{X}, \vec{W})$ such that $(EM, \vec{u}) \models (\vec{Z}' = \vec{z})$ and

(a) $(EM_{\vec{W}=\vec{w}'}, \vec{u}) \models [\vec{X} = \vec{x}']\neg\varphi$ and $s_{\vec{X}=\vec{x}',\vec{W}=\vec{w}'} \succeq s$, where s is the actual world characterized by $(EM, \vec{u})$

(b) $(EM_{\vec{W}=\vec{w}'}, \vec{u}) \models [\vec{X} = \vec{x}, \vec{Z}' = \vec{z}]\varphi$ for all $\vec{Z}' \subseteq \vec{Z}$.

AC3$^{\mathbf{EM}}$. $\vec{X}$ is minimal; no proper subset of $\vec{X}$ satisfies conditions **AC1$^{\mathbf{EM}}$** and **AC2$^{\mathbf{EM}}$**

The difference to the definition of actual causality is in **AC2$^{\mathbf{EM}}$(a)**. The condition restricts the possibilities for the contingency set such that the resulting world must be at least equally normal to the actual world. [23]

Example 4.4.4. Following Halpern and Pearl's first definition of actual causes in [24], we concluded for Example 4.4.1 that, given their contingency sets, both the car and the witness were actual causes that led to the pedestrian's survival. For both $H = 0$, and $W = 1$, the witness world is $\{H = 1, W = 0, PS = 0\}$. If we now assume that drivers do not normally speed and that this behavior is fundamentally immoral, we can say that $\{H = 0, W = 0, PS = 1\}$ is more normal than the actual world $(H = 0, W = 1, PS = 1\}$. What we cannot say about, however, is whether the witness world $\{H = 1, W = 0, PS = 0\}$ or the actual world $(H = 0, W = 1, PS = 1\}$ is more normal. To describe this, unlike Halpern's definition in [21], the normality order is not a total order but a partial order. These two worlds are incomparable and thus do not satisfy the condition **AC2$^{\mathbf{EM}}$(a)**. Thus, according to the definition of Halpern and Hitchcock [23], one can say that in any case $W = 1$ was not an actual cause of the survival of the pedestrian, since there is no witness world more normal than the actual world. This is very intuitive, since in retrospect there was no life-threatening reason at all for the pedestrian to retreat, and thus the witness could not have been the cause.

As mentioned above, $H = 0$ also has only the same witness world $\{H = 1, W = 0, PS = 0\}$. Therefore also $H = 0$ is not a cause for $PS = 1$ because this witness world is not comparable with the actual world. In other words, according to **AC2$^{\mathbf{EM}}$(a)**, the car is not an actual cause for the survival of the pedestrian either. However, this result is not counterintuitive to Halpern and Hitchcock, probably because the car could not have hit the pedestrian in the actual world, since the pedestrian was pulled back by the witness. [23]

On the one hand, this result now leads to a partial solution for the problem of isomorphism, however, there is another similar problem to Bogus Prevention, namely Bogus Permission and this cannot be solved by the definition of actual cause with extended causal models. In the following section we will discuss the problem of Bogus Permission in more detail and explain where the definition of Halpern and Hitchcock fails. [44]

4.4.3 Bogus Permission

Example 4.4.5. Suppose the pedestrian from Example 4.4.1 has got serious injuries due to the unnecessary help of the witness and has to be taken to the hospital. There are two doctors Andrew and Bryan. Andrew gets the task to put the patient on the

drip immediately. Bryan, on the other hand, should perform a plasma exchange. Bryan doesn't know that the plasma exchange has a negative effect on the patient's health, especially if Andrew does his task before. Unfortunately, Andrew does not put his patient on the drip and Bryan unwittingly does what he was told to do. [44]

The causal model for Example 4.4.5 is as follows:

$$\begin{aligned} A &: 1 \text{ if Andrew puts the patient on the drip, 0 if not} \\ B &: 1 \text{ if Bryan performs a plasma exchange, 0 if not} \\ PD &: 1 \text{ if the patient dies, 0 if not} \\ \mathcal{E} &: PD = \neg A \vee B \\ \text{actual World} &: \{A = 0, B = 1, PD = 1\} \end{aligned}$$

By the time Bryan performs a plasma exchange on the patient, the patient's death is already certain, so $B = 1$ should not be the cause of the patient's death, since Bryan was given a bogus permission, since the decisions prior to his decision were already determinant. $A = 1$ should be the cause, since Andrew's inaction has reduced the patient's chances of living to 0. Unfortunately, $\mathbf{AC2^{EM}}$ cannot represent this fact. If we take the same idea from Halpern and Hitchcock and apply it to this example, $\{A = 1, B = 0, PD = 0\}$, the only witness world for both $A = 0$ and $B = 1$, is not comparable to the actual world $\{A = 0, B = 1, PD = 1\}$. Thus, we come to the same conclusion as with the bogus prevention, that both $A = 0$ and $B = 1$ cannot be the cause, since they have the same witness world and it is not comparable to the actual world. This result is counterintuitive since Andrew obviously did not do his job and he should be the cause of the consequence. [44]

In the next section, we explain Zhu's definition which can handle the problem of isomorphism and therefore solve bogus prevention and bogus permission. [44]

4.5 New Definition of Actual Causes appealing to the Default World

Zhu's new definition uses Menzies's definition in [34] of *default worlds*. Default worlds are worlds from a set of worlds defined by their models. Their exogenous variables are set to the default values they would have naturally, if nothing would influence them. [44]

Definition 4.5.1 (Default World by Menzies) [34]. A causal model $M = (\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{E})$ creates default worlds D such that the following conditions hold for $d \in D$:

(a) The exogenous variables $\mathcal{U}$ of d are set to default values.

(b) The endogenous variables $\mathcal{V}$ of d changed according to the structural equations $\mathcal{E}$ without any actor changing anything about them.

The proximity to Halpern and Hitchcock's normality order is easy to see. Thus, Zhu defined default worlds with extended causal models:

Definition 4.5.2 (Default World by Extended Model) [44]. An extended causal model $EM = (\mathcal{U}, \mathcal{V}, \mathcal{R}, \mathcal{E}, \succeq)$ creates default worlds D such that $d \succeq s$ holds for $d \in D$ and any world s created by EM.

In other words, this means that a default world must be at least as normal as all other worlds.

With the definition of default worlds by extended models, Zhu then defined actual causality with default worlds in [44]:

Definition 4.5.3 (Actual Cause appealing to the Default World) [44]. Let EM be an extended causal model and let $\vec{u}$ be a value combination for the exogenous variables $\mathcal{U}$. Further, we define $\vec{u}^d$, the context of the default world in which all exogenous variables are set to their default values. Then $\vec{X} = \vec{x}$ is an actual cause of φ in the extended causal setting $(EM, \vec{u})$ if the following three conditions hold:

AC1$^{\mathbf{def}}$. $(EM, \vec{u}) \models (\vec{X} = \vec{x})$ and $(EM, \vec{u}) \models \varphi$

AC2$^{\mathbf{def}}$. There exists a partition $(\vec{Z}, \vec{W})$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $(\vec{x}', \vec{w}') \neq (\vec{x}, \vec{w})$ of the variables in the subpartition $(\vec{X}, \vec{W})$ such that $(EM, \vec{u}^d) \models (\vec{W} = \vec{w}')$ or $(EM, \vec{u}) \models (\vec{W} = \vec{w}')$ and $(EM, \vec{u}^d) \not\models (\vec{X} = \vec{x})$ and $(M, \vec{u}) \models (\vec{Z}' = \vec{z})$ and

- **(a)** $(EM_{\vec{W}=\vec{w}'}, \vec{u}^d) \models [\vec{X} = \vec{x}']\neg\varphi$
- **(b)** $(EM_{\vec{W}=\vec{w}'}, \vec{u}^d) \models [\vec{X} = \vec{x}, \vec{Z}' = \vec{z}]\varphi$ for all $\vec{Z}' \subseteq \vec{Z}$.

AC3$^{\mathbf{def}}$. $\vec{X}$ is minimal; no proper subset of $\vec{X}$ satisfies conditions **AC1$^{\mathbf{def}}$** and **AC2$^{\mathbf{def}}$**

Again, **AC1$^{\mathbf{def}}$** and **AC3$^{\mathbf{def}}$** remain unchanged and only **AC2$^{\mathbf{def}}$** is improved compared to **AC3$^{\mathbf{EM}}$**. The contingency set is further restricted so that it may only consist of events that have either actually happened or would naturally happen in the default world. Furthermore, the latter also applies to the actual cause, since otherwise there would be a naturally occurring cause $\vec{C} = \vec{c}$ that triggers the actual cause $\vec{X} = \vec{x}$ and would thus itself become the actual cause for φ.

If we look again at Examples 4.2.1, 4.4.1 and 4.4.5, we can see why Definition 4.5.3, is superior to the previous ones:

Example 4.5.4. In Example 4.2.1, we wanted both the lightning and the arsonist's match to be actual causes of the forest fire. Given the default world $\{L = 0, M = 0, FF = 0\}$ and the contingency set $\{M = 0\}$, the condition **AC2$^{\mathbf{def}}$(a)**, $(EM_{M=0}, \vec{u}^d) \models [L = 0]\neg(FF = 1)$ and the condition **AC2$^{\mathbf{def}}$(b)**, $(EM_{M=0}, \vec{u}^d) \models [L = 1](FF = 1)$ are satisfied. Also the conditions **AC1$^{\mathbf{def}}$** and **AC3$^{\mathbf{def}}$** are `true`, these can be taken from Example 4.2.1. Thus, the lightning is an actual cause of the forest fire in any case. For the match of the arsonist this is also `true` if we choose $\{L = 0\}$ as contingency set.

Example 4.5.5. In Example 4.4.1 the witness was actually not an actual cause for the survival of the pedestrian. To show this we use the default world $\{H = 0, W = 0, PS = 1\}$ and every possible contingency set. The two possible contingency sets are $\{H = 0\}$ and $\{\}$. If we test the contingency set $\{H = 0\}$ we see that **AC2$^{\text{def}}$(a)**, $(EM_{H=0}, \vec{u}^d) \models [W = 0]\neg(PS = 1)$ does not hold. The contingency set $\{\}$ does not change the result either, since **AC2$^{\text{def}}$(a)**, $(EM_{\emptyset=\emptyset}, \vec{u}^d) \models [W = 0]\neg(PS = 1)$ is not `true` either. Therefore, there is no partition $(\vec{Z}, \vec{W})$, such that **AC2$^{\text{def}}$** holds and thus $W = 1$ is not an actual cause.

Conversely, the stopping of the car $H = 0$ remains an actual cause for the survival of the pedestrian. This is proved by finding the appropriate contingency set. With $\{W = 0\}$, both **AC2$^{\text{def}}$(a)**, $(EM_{W=0}, \vec{u}^d) \models [H = 1]\neg(PS = 1)$ and the condition **AC2$^{\text{def}}$(b)**, $(EM_{W=0}, \vec{u}^d) \models [H = 0](PS = 1)$ are satisfied. **AC1$^{\text{def}}$** and **AC3$^{\text{def}}$** are also satisfied and so the stopping of the car is solely responsible for the survival of the pedestrian.

Example 4.5.6. Regarding the last Example 4.4.5, Bryan, the doctor who could not prevent the death of the patient, after the misbehavior of his colleague Andrew, should not be an actual cause. Since the two Examples 4.4.1 and 4.4.5 are isomorphic we come to the same conclusion here as in the example before: Bryan is not the actual cause.

The same is `true` for Andrew, of course. By the contingency set $\{B = 0\}$ for which $(EM, \vec{u}^d) \models (B = 0)$ holds, $A = 0$ is the actual cause of the death of the patient.

4.6 Degree of Responsibility

So far, we have only discussed causality. It was about whether an event is a cause for another event. It is not distinguished whether it is only a subsidiary cause or the main cause. The *degree of responsibility*, in short the *responsibility*, is about measuring causes in order to be able to say how much an event is a cause for another event.

Definition 4.6.1 (Degree of Responsibility) [13]. Consider an actual cause $\vec{X} = \vec{x}$ for φ in the causal setting $(EM, \vec{u})$. It follows from the definition of actual causes that there also exists a partition $(\vec{Z}, \vec{W})$ with a setting $(\vec{x}', \vec{w}')$ such that **AC2$^{\text{def}}$** holds. Under the following conditions

(a) each variable in $\vec{W}$ has a different value in $\vec{w}'$ than it has in the actual world

(b) there is no other partition $(\vec{Z}', \vec{W}')$ and setting $(\vec{z}'', \vec{w}'')$ satisfying **AC2$^{\text{def}}$** such that $|\vec{W}'| < |\vec{W}|$ and each variable in $\vec{W}'$ still has a different value in $\vec{w}''$ than it has in the actual world

the degree of responsibility ρ of the cause $\vec{X} = \vec{x}$ is

$$\rho((EM, \vec{u}), (\vec{X} = \vec{x}), \varphi) = \frac{1}{|\vec{W}| + 1}$$

On the other hand, $\rho((EM, \vec{u}), (\vec{X} = \vec{x}), \varphi) = 0$ if $\vec{X} = \vec{x}$ is not a cause for φ.

Example 4.6.2. Consider 11 voters in an election, which have to choose between candidate A or candidate B. If all 11 voters choose candidate A, the individual voters are less responsible for candidate A's electoral victory than if only 6 voters chose candidate A in the election. In both cases, all voters who chose candidate A are actual causes of candidate A's electoral victory. However, their degree of responsibility differs depending on how many voters choose candidate A. In the first case, individual voters may even change their minds and would make no difference in the election outcome. In the second case, however, every vote counts and a single voter could turn the election result. [33]

In [13], Chockler and Halpern further defined the notion of the *degree of blame*, in short *blame*. In addition to the degree of responsibility, the issue here is the degree to which one can blame someone else for one's actions.Intuitively, an action of a person who did not know the consequences of the action cannot be blamed, even if the action was responsible for the outcome. However, since this would go too far, we will not discuss it further in this paper. [13]

4.7 Complexity

The complexity of computing the degree of responsibility depends strongly on the complexity of causality. In [16] and [17], Eiter and Lukasiewicz showed that the problem of deciding whether $\vec{X} = \vec{x}$ is an actual cause of ϕ is generally $\sum_2$P-complete for general recursive models. If a binary model is used, that is, a model where each variable can take exactly two values, it is NP-complete [17]. The same gap occurs in the complexity of the degree of responsibility: Chockler et al. showed in [14] that computing the degree of responsibility for binary models is $FP^{\mathrm{NP}[log\ n]}$-complete. Then, in [13], Chockler and Halpern presented results for general recursive models and proved that the computation of the degree of responsibility is $FP^{\sum_2 \mathrm{P}[log\ n]}$-complete. [13]

CHAPTER 5

Score-based Explanations based on Shapley Values

In order to be able to give the explanations mentioned in Chapter 3, a measure is needed with which the responsibility can be measured. Scores are well suited for this purpose. In this and the next chapter, we will look more closely at scores for explanations, discuss definitions, and examine their complexity.

Using the Shapley Values, Lundberg and Lee [32] defined SHAP (SHapley Additive exPlanations) – an explanatory value that, in the field of machine learning, provides explanations for the results of classification models. In the area of data management, SHAP measures the contribution of tuples in a query answer. To do this, they didn't use the definitions of causality and responsibility, but the cooperative game theory of L. S. Shapley from 1953 [40]. [9]

5.1 Definition of Shapley Values

Let us consider a cooperative game played by players $\mathcal{P}$. The players are divided into teams of variable size, so each possible team is an element of the powerset of players $P(\mathcal{P})$. Now, one can measure how well each team plays. The Game function or wealth function $\mathcal{G} : P(\mathcal{P}) \to \mathbb{R}$ assigns a real value to each team of players according to their game play. Shapley [40] wanted to know how much an individual player contributes to the game and thus defined the *Shapley values*. [31]

Definition 5.1.1 (Shapley Values) [40]. Let p be a player from a set of players $\mathcal{P}$ in a cooperative game with wealth function $\mathcal{G}$. The contribution of p to $\mathcal{G}$ can then be computed as follows:

$$\text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T))$$

Note that by selecting a player p, a team T can be represented as a permutation π of all players $\mathcal{P}$. By the fixed total order of π, player p determines an index in π. A team then consists of all players that come before the player p in the order of π. For example the permutation $[C, B, A]$ and the observed player B, $\pi^{<B}$ describes the team $\{C\}$ and $\pi^{\leq B}$ describes the team $\{C, B\}$.

In Definition 5.1.1 $\frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}$ is used to determine the ratio of teams in which p plays, to all possible teams. To put it differently, the numerator contains the number of all possible permutations of players from T multiplied by the number of all possible permutations from the remaining players without player p and the denominator contains the number of all possible permutations from all players $\mathcal{P}$. This is obvious since we want to see the difference of $\mathcal{G}$ that arises when we consider each team T in any order once with and once without player p. The rest of the players $\mathcal{P} - T - 1$ is only important to us, because there are also different orders and we include every possible permutation in the denominator. $(\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T))$ calculates how much wealth player p adds to team T. The sum adds up these individual contributions and thus calculates a total contribution of the player p over all possible teams. In other words, how much does $\mathcal{G}$ change if only the team up to and excluding player p and the team up to and including player p are chosen, when an arbitrary order is used to sort all players? [8, 9, 31]

Example 5.1.2. Consider a cooperative computer game $\mathcal{G}$ that can be played as a team with any number of players and interpreted as a function $\mathcal{G} : T \to v$ where team T, a subset of all players $\mathcal{P}$, scores v points. Aaron, Bailey and Carlo play this game together and score 100 points. Now Bailey wants to know how much she contributed to the 100 points and lets play any combination of the three friends:

$$\begin{array}{llll} \mathcal{G}(\emptyset) = 0 & \mathcal{G}(\{A\}) = 80 & \mathcal{G}(\{A, B\}) = 180 & \mathcal{G}(\{A, B, C\}) = 100 \\ & \mathcal{G}(\{B\}) = 100 & \mathcal{G}(\{B, C\}) = 150 & \\ & \mathcal{G}(\{C\}) = 50 & \mathcal{G}(\{A, C\}) = 0 & \end{array}$$

However, Bailey notices that whenever Aaron and Carlo play, the two men do not cooperate with each other, and hinder each other from scoring more points. Therefore, Bailey wants to know for sure and puts the results into the formula for Shapley values and notices that she is responsible for the full 100 points:

$$\begin{aligned} &\text{Shapley}_{\mathcal{G}}(\{A, B, C\}, B) = \\ &\frac{0!2!}{3!}(100 - 0) + \frac{1!1!}{3!}(180 - 80) + \frac{1!1!}{3!}(150 - 50) + \frac{2!0!}{3!}(100 - 0) = \\ &\frac{2}{6}100 + \frac{1}{6}100 + \frac{1}{6}100 + \frac{2}{6}100 = 100 \end{aligned}$$

If we calculate the contribution of the other two players, we quickly see that the combi-

nation of Aaron and Carlo is not good and together they do not add any wealth:

$$\text{Shapley}_{\mathcal{G}}(\{A,B,C\},A) =$$
$$\frac{0!2!}{3!}(80-0)+\frac{1!1!}{3!}(180-100)+\frac{1!1!}{3!}(0-50)+\frac{2!0!}{3!}(100-150) =$$
$$\frac{2}{6}80+\frac{1}{6}80-\frac{1}{6}50-\frac{2}{6}50=15$$

$$\text{Shapley}_{\mathcal{G}}(\{A,B,C\},C) =$$
$$\frac{0!2!}{3!}(50-0)+\frac{1!1!}{3!}(0-80)+\frac{1!1!}{3!}(150-100)+\frac{2!0!}{3!}(100-180) =$$
$$\frac{2}{6}50-\frac{1}{6}80+\frac{1}{6}50-\frac{2}{6}80=-15$$

5.2 Properties of Shapley Values

Consider a player p and a team T from the set of remaining players $\mathcal{P}-\{p\}$. The number of all permutations of $\mathcal{P}$ with the following three conditions

- the players from T are in the first $|T|$ positions
- the player p is at position $|T|+1$
- the remaining players are in the last $|\mathcal{P}|-|T|-1$ positions

is equal to the number of permutations of players $|T|$ multiplied by the number of all permutations of the remaining players $(|\mathcal{P}|-|T|-1)$. If we now vary the players in team T, but leave the size of the team the same, we get all combinations of players $\mathcal{P}$ with p at position $|T|+1$. If we now also vary the size of the team, we get all permutations of $\mathcal{P}$, since now the player p can also be at any position of the permutation.

Formally this means:

$$\sum_{T\subseteq\mathcal{P}\setminus\{p\}}|T|!(|\mathcal{P}|-|T|-1)!=|\mathcal{P}|!$$

therefore also

$$\sum_{T\subseteq\mathcal{P}\setminus\{p\}}\frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}=1$$

holds. The latter can be seen in particular in Example 5.1.2. The coefficients of the players' individual contributions $\frac{2}{6}$, $\frac{1}{6}$, $\frac{1}{6}$ and $\frac{2}{6}$ sum to 1.

In addition, the Shapley values have some more properties as well:

- *Efficiency*: The sum of the contributions of all players is the same as the difference between the value of the wealth function of all players and no players:

$$\sum_{p \in \mathcal{P}} \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = \mathcal{G}(\mathcal{P}) - \mathcal{G}(\emptyset) \tag{5.1}$$

Proof 5.2.1 [5]:

$$\begin{aligned}
&\sum_{p \in \mathcal{P}} \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = \\
&= \sum_{p \in \mathcal{P}} \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T)) \\
&= \sum_{p \in \mathcal{P}} \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} \mathcal{G}(T \cup \{p\}) - \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} \mathcal{G}(T)
\end{aligned}$$

Let X be an arbitrary subset of players from all players $\mathcal{P}$, $X \neq \emptyset$ and $X \neq \mathcal{P}$. To determine the coefficients $\frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}$ of $\mathcal{G}(X)$ in the above sum, we make a case distinction: either $X = T \cup \{p\}$ or $X = T$ holds. This gives us the following coefficients:

$$\begin{aligned}
p \in X &: \frac{(|X| - 1)!(|\mathcal{P}| - |X|)!}{|\mathcal{P}|!} \mathcal{G}(X) \\
p \notin X &: -\frac{|X|!(|\mathcal{P}| - |X| - 1)!}{|\mathcal{P}|!} \mathcal{G}(X)
\end{aligned}$$

Summing over all players $\mathcal{P}$, the former coefficient appears exactly $|X|$ times, since p can be exactly $|X|$ different players. For the other coefficient $p \in P \setminus X$ holds, so it appears $|P| - |X|$ times. Therefore we conclude:

$$\begin{aligned}
|X| \frac{(|X| - 1)!(|\mathcal{P}| - |X|)!}{|\mathcal{P}|!} \mathcal{G}(X) - (|\mathcal{P}| - |X|) \frac{|X|!(|\mathcal{P}| - |X| - 1)!}{|\mathcal{P}|!} \mathcal{G}(X) \\
\frac{|X|!(|\mathcal{P}| - |X|)!}{|\mathcal{P}|!} \mathcal{G}(X) - \frac{|X|!(|\mathcal{P}| - |X|)!}{|\mathcal{P}|!} \mathcal{G}(X) = 0
\end{aligned}$$

So all terms, except those for $X = \emptyset$ and $X = \mathcal{P}$ cancel each other out. Since the coefficient for $\mathcal{G}(\emptyset)$ exists only if $T = \emptyset$ and for $\mathcal{G}(\mathcal{P})$ only if $T = \mathcal{P} \setminus \{p\}$, we

conclude:

$$\begin{aligned}\sum_{p\in\mathcal{P}} \text{Shapley}_{\mathcal{G}}(\mathcal{P},p) =& \sum_{p\in\mathcal{P}} \sum_{T\subseteq\mathcal{P}\setminus\{p\}} \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{p\})-\mathcal{G}(T)) \\ =& \sum_{p\in\mathcal{P}} \frac{|(\mathcal{P}\setminus\{p\})|!(|\mathcal{P}|-|(\mathcal{P}\setminus\{p\})|-1)!}{|\mathcal{P}|!}\mathcal{G}(\mathcal{P}) \\ & - \sum_{p\in\mathcal{P}} \frac{|\emptyset|!(|\mathcal{P}|-|\emptyset|-1)!}{|\mathcal{P}|!}\mathcal{G}(\emptyset) \\ =& |\mathcal{P}|\frac{(|\mathcal{P}|-1)!}{|\mathcal{P}|!}(\mathcal{G}(\mathcal{P})-\mathcal{G}(\emptyset)) \\ =& \mathcal{G}(\mathcal{P})-\mathcal{G}(\emptyset)\end{aligned}$$

■

- *Symmetry*: If two players $p, q \in \mathcal{P}$ contribute the same wealth to any possible team, then they should have the same Shapley value:

  $\mathcal{G}(T\cup\{p\}) = \mathcal{G}(T\cup\{q\})$ for all $T \subseteq \mathcal{P}\setminus\{p,q\} \Rightarrow \text{Shapley}_{\mathcal{G}}(\mathcal{P},p) = \text{Shapley}_{\mathcal{G}}(\mathcal{P},q)$

  Proof 5.2.2:

  $$\text{Shapley}_{\mathcal{G}}(\mathcal{P},p) = \sum_{T\subseteq\mathcal{P}\setminus\{p\}} \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{p\})-\mathcal{G}(T))$$

  We make a case distinction for the two cases $q \in T$ and $q \notin T$:

  $$\begin{aligned} q\in T: & \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{p\})-\mathcal{G}(T)) = \\ & = \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{q\}\cup\{p\})-\mathcal{G}(T\cup\{q\})) = \\ & = \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{p\}\cup\{q\})-\mathcal{G}(T\cup\{p\})) \\ q\notin T: & \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{p\})-\mathcal{G}(T)) = \\ & = \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{q\})-\mathcal{G}(T)) \end{aligned}$$

  Since p cannot be a part of T, we can unify both cases back into one:

  $$\sum_{T\subseteq\mathcal{P}\setminus\{q\}} \frac{|T|!(|\mathcal{P}|-|T|-1)!}{|\mathcal{P}|!}(\mathcal{G}(T\cup\{q\})-\mathcal{G}(T))$$

It should be noted that all cases are covered. The first case deals with all teams containing p and the second with all teams not containing p. We draw the following conclusion:

$$\text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = \sum_{T \subseteq \mathcal{P} \setminus \{q\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{q\}) - \mathcal{G}(T)) = \text{Shapley}_{\mathcal{G}}(\mathcal{P}, q)$$

■

- *Dummy*: The contribution and therefore the Shapley value of a player $p \in \mathcal{P}$, which does not add any wealth to any team, is 0.

$$\mathcal{G}(T) = \mathcal{G}(T \cup \{p\}) \text{ for all } T \subseteq \mathcal{P} \setminus \{p\} \Rightarrow \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = 0$$

Proof 5.2.3 [5]:

$$\text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) = \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T))$$

Using $\mathcal{G}(T) = \mathcal{G}(T \cup \{p\})$ for all $T \subseteq \mathcal{P} \setminus \{p\}$, we can follow:

$$\begin{aligned} \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) &= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T \cup \{p\})) \\ &= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} 0 \\ &= 0 \end{aligned}$$

■

- *Additivity*: The contribution of a player $p \in \mathcal{P}$ to the tournament $\mathcal{T}$ consisting of 2 games $\mathcal{G}$ and $\mathcal{H}$, shall be equal to the sum of the contributions to the individual games:

$$\text{Shapley}_{\mathcal{T}}(\mathcal{P}, p) = \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) + \text{Shapley}_{\mathcal{H}}(\mathcal{P}, p)$$

Proof 5.2.4 [5]:

$$
\begin{aligned}
&\text{Shapley}_{\mathcal{T}}(\mathcal{P}, p) = \\
&= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{T}(T \cup \{p\}) - \mathcal{T}(T)) \\
&= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} ((\mathcal{G}(T \cup \{p\}) + \mathcal{H}(T \cup \{p\})) - (\mathcal{G}(T) + \mathcal{H}(T))) \\
&= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) + \mathcal{H}(T \cup \{p\}) - \mathcal{G}(T) - \mathcal{H}(T)) \\
&= \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{G}(T \cup \{p\}) - \mathcal{G}(T)) \\
&\qquad + \sum_{T \subseteq \mathcal{P} \setminus \{p\}} \frac{|T|!(|\mathcal{P}| - |T| - 1)!}{|\mathcal{P}|!} (\mathcal{H}(T \cup \{p\}) - \mathcal{H}(T)) \\
&= \text{Shapley}_{\mathcal{G}}(\mathcal{P}, p) + \text{Shapley}_{\mathcal{H}}(\mathcal{P}, p)
\end{aligned}
$$

■

To be able to give explanations in arbitrary domains, such as machine learning or databases, all that is missing is the definition of the function $\mathcal{G}$. In the following two sections, we want to show how the function $\mathcal{G}$ is defined for the mentioned domains and how the Shapley Values can be used to give explanations.

5.3 Shapley Values in Machine Learning

Lundberg and Lee [32] defined *SHAP*, a score in the area of machine learning to explain how much individual features contribute to the outcome of an entity. For this purpose they used the Shapley values. The players of the Shapley values are an analogy to the features of the input entities of the machine learning classifier $\mathcal{C}$. In this way, the only thing missing is a matching wealth function to calculate the contribution of each feature f of all features $\mathcal{F}$. But exactly here lies the problem, because a game, which is played by only a real subset of the players, makes no sense with features. A classifier can only use whole entities as input. For a partial entity, the classifier cannot return an outcome. Therefore SHAP uses a probability distribution Pr. [15]

We define an expected value for SHAP as follows:

Definition 5.3.1 (Expected Value) [15]. The expected classification outcome of a classifier $\mathcal{C}$ for a complete instance $e \in \mathcal{E}$ and distribution Pr is $E_{Pr}[\mathcal{C}|\{e\}] = \mathcal{C}(e)$. Furthermore, we define a set of entities $\mathbf{e}_F$ in respect to entity $\mathbf{e}$ and features $F \subseteq \mathcal{F}$:

$$\mathbf{e}_F = \{e \in \mathcal{E} \mid f(e) = f(\mathbf{e}) \text{ if } f \in F\}$$

It follows, that $\mathbf{e}_\emptyset = \mathcal{E}$ and $\mathbf{e}_\mathcal{F} = \{\mathbf{e}\}$. The expected value for a partial entity $\pi_\mathbf{e}$ is determined by its corresponding set of entities $\mathbf{e}_F$, and is therefore

$$E_{Pr}[\mathcal{C}|\{\pi_\mathbf{e}\}] = E_{Pr}[\mathcal{C}|\mathbf{e}_F] = \sum_{e \in \mathcal{E}} \mathcal{C}(e) Pr(e|\mathbf{e}_F) \tag{5.2}$$

That is why the wealth function $\mathcal{G}$ for the SHAP score is defined as follows:

$$\mathcal{G}(F) = E_{Pr}[\mathcal{C}|\mathbf{e}_F] \tag{5.3}$$

From this, using the Shapley values, the SHAP score for a single feature is obtained by calculating the contribution each time in a different context of features:

Definition 5.3.2 (SHAP) [15]. Let $\mathcal{C}$ be a classifier and Pr be a data distribution, then the SHAP score of a feature $f \in \mathcal{F}$ of an entity $\mathbf{e} \in \mathcal{E}$ is:

$$\text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f) = \sum_{F \subseteq \mathcal{F} \setminus \{f\}} \frac{|F|!(|\mathcal{F}| - |F| - 1)!}{|\mathcal{F}|!} (E_{Pr}[\mathcal{C}|\mathbf{e}_{F \cup \{f\}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_F])$$

One can check that the properties described in Section 5.2 also apply to SHAP.

5.3.1 Complexity

The interesting aspect of the SHAP score is that it is solvable in polynomial time under certain conditions. In this section we want to take a closer look at the complexity of the SHAP score and establish conditions which lead to the fact that the SHAP score is tractable. To prove this, we first define the following problem:

Functional SHAP problem [15]: Given a data distribution Pr and a function $\mathcal{C}$. The *functional SHAP problem*, denoted F-SHAP$(\mathcal{C}, Pr)$ is to compute

$$\text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f_1), \ldots, \text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f_n).$$

SHAP over Fully-Factorized Distributions

The SHAP score is all about the expected value, which arises from the used distribution Pr. In the following, we will look specifically at SHAP explanations under fully-factorized distributions. Since the function $\mathcal{C}$ in Definition 5.3.2 often arises from supervised learning, we can assume that the data is fully-factorized, which also simplifies the estimation. Furthermore, a fully-factorized distribution helps because computing SHAP explanations can be tractable for several popular classifiers.

Fully-Factorized Expectation Problem [15]: Given a fully-factorized probability distribution Pr and a function $\mathcal{C} : \mathcal{E} \to \mathbb{R}$. The *fully-factorized expectation problem*, denoted $E_{Pr}(\mathcal{C})$, is to calculate $E_{Pr}[\mathcal{C}|\mathcal{E}]$.

Now we want to show that there is a connection between the calculation of expectations and the calculation of SHAP explanations:

Proof 5.3.3 [15]: We can adopt Equation 5.1, the efficiency of Shapley values, for SHAP

$$\sum_{f \in \mathcal{F}} \text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f) = E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_{\emptyset}\}] = \mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathcal{E}],$$

It follows immediately, that $E_{Pr}(\mathcal{C}) \leq^P \text{F-SHAP}(\mathcal{C}, Pr)$ holds, since

$$E_{Pr}[\mathcal{C}|\mathcal{E}] = \mathcal{C}(\mathbf{e}) - \sum_{f \in \mathcal{F}} \text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f).$$

In other words, given a function $\mathcal{C} : \mathcal{E} \to \mathbb{R}$ and an oracle to calculate $\text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f)$ for each $f \in \mathcal{F}$, we can calculate the expected value $E_{Pr}[\mathcal{C}|\mathcal{E}]$. In the following we will show that also the converse, that is $\text{F-SHAP}(\mathcal{C}, Pr) \leq^P E_{Pr}(\mathcal{C})$, holds. It then follows:

Theorem 5.3.4 [15]. For any function $\mathcal{C} : \{0,1\}^n \to \mathbb{R}$, $\text{F-SHAP}(\mathcal{C}, Pr) \equiv^P E_{Pr}(\mathcal{C})$ holds.

This means that the complexity of computing an expected value $E_{Pr}[\mathcal{C}|\mathcal{E}]$ of a fully-factorized distribution Pr is equal to the complexity of computing SHAP scores for each feature of an entity.

W.l.o.g., we choose an $n+1$-dimensional binary entity $e \in \mathcal{E}$, for which we want to explain the outcome $\mathcal{C}(e)$ in the following. Further, we define the fully-factorized distribution Pr as $n+1$ rational numbers $p_i = Pr(X_i = 1)$ for all $i \in \{0, \ldots, n\}$. This gives $Pr(X_i = 0) = 1 - p_i$. It follows

$$\text{SHAP}_{\mathcal{C},Pr}(\mathbf{e}, f) = \sum_{k=0}^{n} \frac{k!(n-k)!}{(n+1)!} D_k, \tag{5.4}$$

where

$$D_k = \sum_{F \in \{F' \subseteq \mathcal{F} || F'| = k\}} (E_{Pr}[\mathcal{C}|\mathbf{e}_{F \cup \{0\}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_F])$$

Now we want to simplify D_k further. To do this, we split the function in $\mathcal{C}$ in $n+1$ binary features, into two functions $\mathcal{C}_0$ and $\mathcal{C}_1$ in n binary features each:

$$\mathcal{C}(e) = \begin{cases} \mathcal{C}_0(e) & \text{if } f_0(e) = 0 \\ \mathcal{C}_1(e) & \text{if } f_0(e) = 1 \end{cases}$$

Thus we can conclude that

$$E_{Pr}[\mathcal{C}|\mathbf{e}_{F \cup \{0\}}] = E_{Pr}[\mathcal{C}_1|\mathbf{e}_F]$$
$$E_{Pr}[\mathcal{C}|\mathbf{e}_F] = E_{Pr}[\mathcal{C}_0|\mathbf{e}_F] * (1 - p_0) + E_{Pr}[\mathcal{C}_1|\mathbf{e}_F] * p_0$$

and therefore D_k can also be defined as

$$D_k = (1 - p_0) \sum_{F \subseteq \mathcal{F}, |F| = k} (E_{Pr}[\mathcal{C}_1|\mathbf{e}_F] - E_{Pr}[\mathcal{C}_0|\mathbf{e}_F])$$

Any oracle for $E_{Pr}(\mathcal{C})$ is also an oracle for $E_{Pr}(\mathcal{C}_0)$ and $E_{Pr}(\mathcal{C}_1)$, since only $p_0 = 0$ or $p_0 = 1$ has to be set respectively. Thus it is only necessary to show that

$$\sum_{F\subseteq\{0,\ldots,n\},|F|=k} E_{Pr}[\mathcal{C}|\mathbf{e}_F]$$

can be computed for all $k \in 1,\ldots,n$ with only $n+1$ calls to the oracle $E_{Pr}(\mathcal{C})$. This would proof, that all D_k can be computed in polynomial time and thus by Equation 5.4 would also prove Theorem 5.3.4.

Consider a real number $z > 0$ and the distribution $Pr^z(X_i) = \frac{p_i+z}{1+z}$, for all $i \in \{1,\ldots,n\}$. Because of independence, the probabilities for instance e are:

$$Pr(e) = \prod_{i\in\{1,\ldots,n\},X_i=1} p_i \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} 1-p_i$$

$$Pr^z(e) = \prod_{i\in\{1,\ldots,n\},X_i=1} \frac{p_i+z}{1+z} \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} 1-\frac{p_i+z}{1+z}$$

This yields the following equality:

$$Pr(e) = \prod_{i\in\{1,\ldots,n\},X_i=1} p_i \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} 1-p_i$$

$$Pr(e) \prod_{i\in\{1,\ldots,n\},X_i=1} (1+\frac{z}{p_i}) = \prod_{i\in\{1,\ldots,n\},X_i=1} p_i + z \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} 1-p_i$$

$$Pr(e) \prod_{i\in\{1,\ldots,n\},X_i=1} (1+\frac{z}{p_i}) = (1+z)^n \prod_{i\in\{1,\ldots,n\},X_i=1} \frac{p_i+z}{1+z} \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} \frac{1+z-p_i-z}{1+z}$$

$$Pr(e) \prod_{i\in\{1,\ldots,n\},X_i=1} (1+\frac{z}{p_i}) = (1+z)^n \prod_{i\in\{1,\ldots,n\},X_i=1} \frac{p_i+z}{1+z} \cdot \prod_{i\in\{1,\ldots,n\},X_i=0} 1-\frac{p_i+z}{1+z}$$

$$Pr(e) \prod_{i\in\{1,\ldots,n\},X_i=1} (1+\frac{z}{p_i}) = (1+z)^n Pr^z(e) \tag{5.5}$$

Moreover, because of Equation 5.2 and the conditional probability $Pr(A|B) = \frac{Pr(A\cap B)}{Pr(B)}$:

$$E_{Pr}[\mathcal{C}|\mathbf{e}_F] = \sum_{e\in\mathcal{E}} \mathcal{C}(e)\frac{Pr(e\cap\mathbf{e}_F)}{Pr(\mathbf{e}_F)}$$

$$E_{Pr}[\mathcal{C}|\mathbf{e}_F] = \sum_{e\in\mathbf{e}_F} \mathcal{C}(e)\frac{Pr(e)}{Pr(\mathbf{e}_F)}$$

$$E_{Pr}[\mathcal{C}|\mathbf{e}_F] = \frac{1}{Pr(\mathbf{e}_F)} \sum_{e\in\mathbf{e}_F} \mathcal{C}(e)Pr(e)$$

and since $Pr(\mathbf{e}_F) = \prod_{i\in F} p_i$ holds, it follows that

$$E_{Pr}[\mathcal{C}|\mathbf{e}_F] = \frac{1}{\prod_{i\in F} p_i} \sum_{e\in\mathbf{e}_F} \mathcal{C}(e)Pr(e). \tag{5.6}$$

By Equation 5.6, the exchange of two sums, the identity $\sum_{S\subseteq A}\prod_{i\in S}u_i = \prod_{i\in A}(1+u_i)$ and Equation 5.5, we conclude.

$$\begin{aligned}
\sum_{k=0}^{n} z^k \sum_{F\subseteq\{0,\dots,n\},|F|=k} E_{Pr}[\mathcal{C}|\mathbf{e}_F] &= \sum_{F\subseteq\{0,\dots,n\},|F|=k} z^{|F|} E_{Pr}[\mathcal{C}|\mathbf{e}_F] \\
&= \sum_{F\subseteq\{0,\dots,n\},|F|=k} \frac{z^{|F|}}{\prod_{i\in F} p_i} \sum_{e\in\mathbf{e}_F} \mathcal{C}(e)Pr(e) \\
&= \sum_{e\in\mathcal{E}} \mathcal{C}(e)Pr(e) \sum_{e\in\mathbf{e}_F} \frac{z^{|e_F|}}{\prod_{i\in e} p_i} \\
&= \sum_{e\in\mathcal{E}} \mathcal{C}(e)Pr(e) \prod_{i\in\{1,\dots,n\},X_i=1} (1+\frac{z}{p_i}) \\
&= (1+z)^n \sum_{e\in\mathcal{E}} \mathcal{C}(e)Pr^z(e) \\
&= (1+z)^n E^z_{Pr}(\mathcal{C})
\end{aligned}$$

By this equality, we can form a linear system of equations with $n+1$ different values for z. Note that we thereby form a non-singular Vandermonde matrix, which has a unique solution that can be solved in polynomial time. Thus we proved that it takes only $n+1$ calls to the oracle $E^z_{Pr}(\mathcal{C})$ to compute F-SHAP$(\mathcal{C}, Pr)$ and thereby proved Theorem 5.3.4. ■

5.4 Shapley Values in Databases

Also in the area of databases Shapley values can be used to determine the contribution of single tuples for boolean query results. It also follows that one can determine which tuples in particular change the query answer of a database. [8]

Again, the players are an analogy to the tuple τ in the database D. The boolean query Q becomes the wealth function $\mathcal{Q}$ which is defined as follows:

Let $T \subseteq D$,

$$\mathcal{Q}(T) = \begin{cases} 1 & \text{if } T \models Q \\ 0 & \text{if } T \not\models Q \end{cases}$$

With this we can calculate the Shapley value of a database tuple:

Definition 5.4.1 (Shapley Values for Database Tuples) [8]. Consider a database D and a wealth function $\mathcal{Q}$ defined by a boolean query Q, then the Shapley value for tuple $\tau \in D$ is:

$$\text{Shapley}_{\mathcal{Q}}(D,\tau) = \sum_{T\subseteq D\setminus\{\tau\}} \frac{|T|!(|D|-|T|-1)!}{|D|!}(\mathcal{Q}(T\cup\{\tau\}) - \mathcal{Q}(T))$$

Livshits et al. investigated in [30] the complexity of computing Shapley values for database tuples. They proved that Shapley values for BCQs without self-joins can be computed in polynomial time if and only if they are hierarchical. Hierarchical in this context means that for every two existential variables x and y

$$Atoms(x) \subseteq Atoms(y) \vee Atoms(y) \subseteq Atoms(x) \vee Atoms(x) \cap Atoms(y) = \emptyset$$

holds. For example, $Q : \exists x \exists y \exists z(R(x, y) \wedge S(x, z))$ is hierarchical because $Atoms(x) = \{R(x, y), S(x, z)\}$ and $Atoms(y) = \{R(x, y)\}$ and $Atoms(z) = \{S(x, z)\}$. On the other hand, $Q : \exists x \exists y(R(x) \wedge S(x, y) \wedge T(y))$ is not hierarchical because $Atoms(x) = \{R(x) \wedge S(x, y)\}$ and $Atoms(y) = \{S(x, y) \wedge T(y)\}$. Even though the proof is different, the criteria used for this tractability are the same ones used for the evaluation of BCQs over probabilistic databases. Furthermore, one can extend the results to summation over CQs using the same criteria, since the expected value applied in Shapley value is linear. However, aggregations, such as *max*, *min*, and *avg*, over non-hierarchical queries are harder. Livshits et al. provide an approximate result for this in [30]. [8]

5.4.1 Banzhaf Power Index

Another score for database tuples,is the *Banzhaf power index*. This one is also hard to compute after [30] and provably #P-hard in general. In addition, it does not have the same previously discussed properties of Shapley values, but is defined very similarly to them:

Definition 5.4.2 (Banzhaf Power Index) [8]. Consider a database D and a wealth function $\mathcal{Q}$ defined by a boolean query Q, then the Banzhaf Power Index for tuple $\tau \in D$ is:

$$\mathrm{BPI}_{\mathcal{Q}}(D, \tau) = \frac{1}{2^{|D|-1}} \sum_{T \subseteq D \setminus \{\tau\}} (\mathcal{Q}(T \cup \{\tau\}) - \mathcal{Q}(T))$$

The individual contributions a player gives to the various teams are not weighted by the number of permutations of the teams as in Shapley values, but by the combinations of the teams. In [30] it is proven that the Banzhaf Power Index matches the causal effect score, which will be mentioned in Section 6.2. [8]

CHAPTER 6

Score-based Explanations based on Causality and Responsibility

Also with causality from Chapter 4 scores for explanations can be defined. In this chapter, we will discuss these in more detail.

6.1 Counterfactual Interventions in Machine Learning

Recall counterfactual interventions from Section 3.1, which are changes to the initial state to achieve the desired result. Formally, we define a counterfactual intervention for a particular entity $e \in \mathcal{E}$ as a pair (f, v) if changing the feature $f \in \mathcal{F}$ to the value $v \in dom(f)$ alone yields the desired change in classification outcome $\mathcal{C}(e)$. [9]

To explain how important a single feature f is for the classification outcome, one could ask how large the expected value is that f with a random value $v \in dom(f)$ forms a counterfactual intervention. One can immediately see that a feature that changes the classification outcome by any change in value is very likely to be the cause of the outcome, and a feature with which the classification outcome cannot be changed, cannot be a cause. [9] This principle is defined by the *COUNTER* score:

Definition 6.1.1 (COUNTER) [9]. Let $\mathcal{C}$ be a classifier and Pr be a data distribution, then the COUNTER score for feature $f \in \mathcal{F}$ and entity $\mathbf{e} \in \mathcal{E}$ is:

$$\text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e}, f) = \mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus\{f\}}]$$

Thus, the COUNTER score gives the expected value for a change in the classification outcome due to a change in the single feature f. [9] In the following example, we calculate the COUNTER score for a specific instance:

Example 6.1.2. Consider a data distribution Pr and a classifier $\mathcal{C}$ for entities with two features A and B resulting in the following labelling:

A \ B	0	1	2	3
0	$\mathcal{C}(\langle 00\rangle) = 1$	$\mathcal{C}(\langle 01\rangle) = 0$	$\mathcal{C}(\langle 02\rangle) = 1$	$\mathcal{C}(\langle 03\rangle) = 1$
1	$\mathcal{C}(\langle 10\rangle) = 1$	$\mathcal{C}(\langle 11\rangle) = 1$	$\mathcal{C}(\langle 12\rangle) = 1$	$\mathcal{C}(\langle 13\rangle) = 0$
2	$\mathcal{C}(\langle 20\rangle) = 1$	$\mathcal{C}(\langle 21\rangle) = 1$	$\mathcal{C}(\langle 22\rangle) = 1$	$\mathcal{C}(\langle 23\rangle) = 0$
3	$\mathcal{C}(\langle 30\rangle) = 1$	$\mathcal{C}(\langle 31\rangle) = 1$	$\mathcal{C}(\langle 32\rangle) = 1$	$\mathcal{C}(\langle 33\rangle) = 0$

Let us further find out from which of the two features A and B the classification outcome $\mathcal{C}(\langle 00\rangle) \neq 0$ depends more. For this we calculate the COUNTER score for both features:

$$\text{COUNTER}_{\mathcal{C},Pr}(\langle 00\rangle, A) = \mathcal{C}(\langle 00\rangle) - E_{Pr}[\mathcal{C}|\{\langle 00\rangle, \langle 10\rangle, \langle 20\rangle, \langle 30\rangle\}] = 1 - \frac{4}{4} = 0$$
$$\text{COUNTER}_{\mathcal{C},Pr}(\langle 00\rangle, B) = \mathcal{C}(\langle 00\rangle) - E_{Pr}[\mathcal{C}|\{\langle 00\rangle, \langle 01\rangle, \langle 02\rangle, \langle 03\rangle\}] = 1 - \frac{3}{4} = \frac{1}{4}$$

In contrast to feature A, feature B can be used to form a counterfactual version of $\langle 00\rangle$, so $\text{COUNTER}_{\mathcal{C},Pr}(\langle 00\rangle, B) > 0$, since this requires a counterfactual intervention. The COUNTER score of B, means that $\frac{1}{4}$ is the expected classification outcome by changing the value of feature B. The COUNTER score of 0 means that any change in the feature does not change the classification outcome.

However, the COUNTER score has a problem: an entity does not always have counterfactual interventions, meaning that changing a single feature does not necessarily change the outcome. Such an entity would then have COUNTER score 0 for each individual feature. Therefore, no explanation could be made about such an entity and the COUNTER score would be useless. To overcome the problem, a score is needed that evaluates the features not only according to possible counterfactual interventions. [8]

In Section 3.2 we presented actual causes together with their contingency sets. Formally, a pair (f, v) is an actual cause for an entity e with contingency set Γ, which is a set of features of e with new values, if (f, v) is a counterfactual intervention for e previously changed by Γ. Thus counterfactual interventions are actual causes with empty contingency set, since nothing about e needs to be changed before it is a counterfactual intervention. The extent to which an actual cause is a counterfactual intervention can be derived from the responsibility of a cause from Section 4.6. This is defined as $\frac{1}{1+|\Gamma|}$. [6] This gives the definition for the *x-RESP* score:

Definition 6.1.3 (x-RESP) [8]. If Γ is a minimum-size contingency set for an actual cause with feature f, then the x-RESP score for f is

$$\text{x-RESP}(f) = \frac{1}{1 + |\Gamma|}$$

The x-RESP score should indicate how responsible a feature is for the classification outcome. The smaller the contingency set, the larger the x-RESP score and thus also the responsibility for the classification outcome. The x-RESP score for a counterfactual intervention, that is, an actual cause with the contingency $\emptyset$, is 1. [6]

Now consider Example 6.1.2 again and let us compute the x-RESP score for features A and B:

Example 6.1.4. In Example 6.1.2, although only $(B, 1)$ is a counterfactual intervention for entity 00, $(A, 1)$, $(A, 2)$ and $(A, 3)$ are at least actual causes with contingency set $\Gamma = \{(B = 3)\}$, so we can compute the following x-RESP scores:

$$\text{x-RESP}(A) = \frac{1}{1 + |\{(B = 3)\}|} = \frac{1}{2}$$
$$\text{x-RESP}(B) = \frac{1}{1 + |\emptyset|} = 1$$

It can be seen immediately that by changing B alone, one can change the classification outcome $\mathcal{C}(\langle 00 \rangle)$ to the value 0. A is only half responsible, because before A can form a counterfactual intervention, the contingency set $\{(B = 3)\}$ of size 1 must be applied.

However, the x-RESP score also causes problems. Take a close look at the table in Example 6.1.2. In order to change the classification outcome with the feature B, it must be set exactly to the value 1. So the classification outcome is very unstable by changing the feature B. This is in contrast to when B is set to the value 3. Actually, it almost doesn't matter what value the feature A is set to as long as the feature B is set to the value 3.

In Example 6.1.2 and Example 6.1.4 it looks like the counterfactual intervention $(B, 1)$ is the cause for the classification outcome. A better explanation, however, would be to set the feature B to the value 3, since there are so many different ways to change the classification outcome. To achieve better explanations and at the same time not neglect the fact that a counterfactual intervention can be formed with the feature B, Bertossi et al. introduced the *RESP* score in [9]. This score computes the ratio between the expected value of the classifier given any change in a feature and the size of the contingency set used. [8] Formally, the RESP score is defined as follows:

Definition 6.1.5 (RESP) [8]. Consider a classifier $\mathcal{C}$ and an entity e with classification outcome $\mathcal{C}(e)$. Let Γ be a contingency set, where the features in Γ must not be set to the original values of e. Further, let $\mathbf{e^*}$ be the entity e modified by Γ with $\mathcal{C}(\mathbf{e^*}) = \mathcal{C}(e)$. Also, there is at least one feature f of $\mathbf{e^*}$ whose change would cause a change in $\mathcal{C}(\mathbf{e^*})$. The RESP score for f with Γ is then defined as follows:

$$\text{RESP}_{\mathcal{C},Pr}(\mathbf{e}, f, \Gamma) = \frac{\mathcal{C}(\mathbf{e^*}) - E_{Pr}[\mathcal{C}|\mathbf{e^*}_{\mathcal{F}\setminus\{f\}}]}{1 + |\Gamma|}$$

$\text{RESP}_{\mathcal{C},Pr}(\mathbf{e}, f)$ is defined as $\max_\Gamma(\text{RESP}_{\mathcal{C},Pr}(\mathbf{e}, f, \Gamma))$ where Γ is the smallest contingency set at which $\text{RESP}_{\mathcal{C},Pr}(\mathbf{e}, f, \Gamma) > 0$ holds.

If we now apply the RESP score to Example 6.1.2, we come to the following conclusions:

Example 6.1.6. In Example 6.1.2, for feature A there are three actual causes $(A,1)$, $(A,2)$ and $(A,3)$ with contingency set $\{(B=3)\}$. This results in

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, A, \{(B=3)\}) = \frac{\mathcal{C}(\langle 03\rangle) - E_{Pr}[\mathcal{C}|\{\langle 03\rangle, \langle 13\rangle, \langle 23\rangle, \langle 33\rangle\}]}{1+|\{(B=3)\}|} = \frac{1-\frac{1}{4}}{2} = \frac{3}{8}$$

Furthermore, since $\Gamma = \{(B=3)\}$ is the only contingency set with $\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, A) > 0$

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, A) = \frac{3}{8}.$$

holds.

However, this looks a little different for feature B. Feature B can form the counterfactual intervention $(B,1)$ as well as the actual cause $(B,3)$ with the three possible contingency sets $\{(A,1)\}$, $\{(A,2)\}$ or $\{(A,3)\}$. This results in the following RESP scores:

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B, \emptyset) = \frac{\mathcal{C}(\langle 00\rangle) - E_{Pr}[\mathcal{C}|\{\langle 00\rangle, \langle 01\rangle, \langle 02\rangle, \langle 03\rangle\}]}{1+|\emptyset|} = \frac{1-\frac{3}{4}}{1} = \frac{1}{4}$$

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B, \{(A=1)\}) = \frac{\mathcal{C}(\langle 10\rangle) - E_{Pr}[\mathcal{C}|\{\langle 10\rangle, \langle 11\rangle, \langle 12\rangle, \langle 13\rangle\}]}{1+|\{(A=1)\}|} = \frac{1-\frac{3}{4}}{2} = \frac{1}{8}$$

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B, \{(A=2)\}) = \frac{\mathcal{C}(\langle 20\rangle) - E_{Pr}[\mathcal{C}|\{\langle 20\rangle, \langle 21\rangle, \langle 22\rangle, \langle 23\rangle\}]}{1+|\{(A=2)\}|} = \frac{1-\frac{3}{4}}{2} = \frac{1}{8}$$

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B, \{(A=3)\}) = \frac{\mathcal{C}(\langle 30\rangle) - E_{Pr}[\mathcal{C}|\{\langle 30\rangle, \langle 31\rangle, \langle 32\rangle, \langle 33\rangle\}]}{1+|\{(A=3)\}|} = \frac{1-\frac{3}{4}}{2} = \frac{1}{8}$$

Since $\Gamma = \emptyset$ is the smallest contingency set, with $\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B, \Gamma) > 0$

$$\text{RESP}_{\mathcal{C},Pr}(\langle 00\rangle, B) = \frac{1}{4}.$$

holds.

In Example 6.1.6 it is now clear that both feature A and feature B are causes for the classification outcome, and changing feature A may even have a little more impact, even if it requires the change of feature B.

6.1.1 Complexity

To better examine and compare the scores, in this section we look at the complexity of the RESP score.

RESP over Fully-Factorized Distributions

Note that the complexity of the RESP score is mainly to calculate the x-RESP score for the same entity. The definition of the RESP score describes that the RESP score can be calculated from the multiplication of the x-RESP score and the COUNTER score for the entity modified with the contingency set.

To calculate the RESP score, the COUNTER score can be calculated first. This is much easier to calculate. From Equation 5.2

$$E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus\{f\}}] = \sum_{e \in \mathbf{e}_{\mathcal{F}\setminus\{f\}}} \mathcal{C}(e)Pr(e)$$

can be derived. This requires only a single scan over $dom(f)$ and at most $|dom(f)|$ calls to the classifier $\mathcal{C}$. [9]

In practice, usually not many features need to be changed to alter the classification outcome. Therefore, the COUNTER score is often non-zero and thus equal to the RESP score. [9]

If the COUNTER score should still be zero, contingency sets of size $1, 2, \ldots$ are examined until a non-zero RESP score is found. First the features for the contingency set are chosen. Then the RESP score is calculated by taking the Cartesian product of the values of the selected contingency set and the domains of the remaining features. [9]

Of course, in the worst case, all values of all features must be tested. This makes the calculation of the RESP score an NP-complete problem. However, since the worst case occurs very rarely in practice, one usually achieves a non-zero RESP score even with contingency sets of small size. We will show this in Chapter 9 on a real world dataset. In Chapter 11 we will then go into more detail on how meaningful larger contingency sets are. [7, 9]

6.2 Counterfactual Interventions in Databases

As in Section 5.4 with the Shapley values, causality can be used to perform database repairs. The goal here is still to determine which of the tuples in a database makes the difference between an inconsistent database to a consistent database. In other words, we are looking for counterfactual interventions that make the inconsistent database consistent by deleting tuples. [8]

6.2.1 COUNTER, x-RESP and RESP in Databases

With databases, there are only two ways to modify instances: Either deleting a tuple or not. Thus, no expected value can be calculated, since the result is always binary. Regardless of whether we delete the database tuple or not, in both cases we can tell exactly whether the database is consistent or inconsistent with respect to Q. Therefore

the COUNTER score of a database tuple can only be either 0 or 1 and is not really meaningful. The only statement that can be made is:

- If the inconsistent database D becomes consistent when tuple $\tau \in D$ is deleted:

$$\text{COUNTER}_Q(D, \tau) = 1$$

- If the inconsistent database D remains inconsistent when tuple $\tau \in D$ is deleted:

$$\text{COUNTER}_Q(D, \tau) = 0$$

The former condition describes $\tau \in D$ as a counterfactual intervention. The second condition, on the other hand, expresses that the deletion of τ has no effect on the consistency of the database D. In practice, there is usually not just a single tuple involved in database repairs, so the $\text{COUNTER}_Q(D, \tau) = 0$ occurs much more frequently. Often it takes multiple tuples to be deleted. The more tuples that are, the less important the single tuple is in the set and vice versa. The idea of x-RESP score from this chapter can be used to form the following definition:

Definition 6.2.1 (x-RESP for Database Repairs) [8, 31]. Given an inconsistent database D and a tuple $\tau \in D$. If there is a minimum-size contingency set Γ for τ in D, then it holds:

$$\text{x-RESP}(\tau) = \frac{1}{1 + |\Gamma|}$$

If there is no contingency set for τ, then: $\text{x-RESP}(\tau) = 0$

If the same idea is applied to the RESP score, it can be seen that it always equals the x-RESP, since the COUNTER score is only either 0 or 1.

It can be noticed that explanations with counterfactual interventions make more sense for classifiers than for database repairs. While there is a helpful score for tuples in database repairs by the x-RESP score, there are additionally the COUNTER and the RESP score for classifiers. For this reason, we will mainly focus on explanations for classifier scores in the following chapter, as these can be compared better.

6.2.2 Causal Effect

Nevertheless, the *causal effect* should be mentioned here. This was introduced in [39] and is very similar to the COUNTER score described in this chapter. A *probabilistic database* D^{Pr} is used, in which each tuple of a database D is uniformly and independently assigned a probability, with tuples that do not appear in the database being assigned probability 0. Then, the expected value that query Q is satisfied can be calculated. The causal effect of tuple τ is the difference of the expected values of query Q as the probability of τ is changed. If the deletion of τ changes much in the expected value, also the causal effect for τ is high. [8]

Definition 6.2.2 (CAUSAL EFFECT) [8, 31]. Given a database D that is inconsistent with respect to the query Q and a tuple $\tau \in D$. Let $D_{\tau=0}$ be the database D with the probability of tuple τ changed to 0. Equivalently, let the probabilistic database $D_{\tau=1}$ be the database D where the probability for tuple τ has been set to 1. The causal effect $\mathrm{CE}_{Q,Pr}(D,\tau)$ is then

$$\mathrm{CE}_{Q,Pr}(D,\tau) = E_{Pr}[Q|D_{\tau=1}] - E_{Pr}[Q|D_{\tau=0}]$$

Example 6.2.3. Consider the database D with its two relations R and S and a denial constraint DC as given below.

R	A	B
	a	b
	a	c
	c	b

S	C
	b
	c

$$DCs: \ \neg\exists x \exists y (R(x,y) \wedge S(y))$$

Table 6.1: An inconsistent database with its denial constraint. [8]

By assigning probabilities uniformly, the database D^{Pr} then looks like this:

R^{Pr}	A	B	Pr
	a	b	$\frac{1}{2}$
	a	c	$\frac{1}{2}$
	c	b	$\frac{1}{2}$

S^{Pr}	C	Pr
	b	$\frac{1}{2}$
	c	$\frac{1}{2}$

Table 6.2: A probabilistic database. [8]

Assuming the probability for the tuple $S^{Pr}(b)$ is set to 1, the reasons that the denial constraint is satisfied and the database is inconsistent are $R(a,b)$, $R(a,c) \wedge S(c)$, and $R(c,b)$. The expected value of such a query is then $E_{Pr}[DC|D_{S(b)=1}] = \frac{13}{16}$, since the probability of all three reasons not occurring is only $\frac{3}{16}$. On the other hand, setting the probability of $S^{Pr}(b)$ to 0, we get $E_{Pr}[DC|D_{S(b)=0}] = \frac{1}{4}$, since then only $R(a,c) \wedge S(c)$ is a cause for DC to be satisfied. By the large difference of the two expected values, the causal effect, it can be seen that the tuple $S(b)$ is an important component to fulfill DC. Accordingly, the deletion of $S(b)$ causes the largest contribution to make the database consistent. [8]

In closing this section, it is worth noting that [10] presents a further advancement in reducing the amount of data that has to be deleted so that an inconsistent database becomes consistent. Bertossi and Salimi focused on database repairs at the attribute level, by setting specific attributes in the database to NULL. By doing so, they achieved consistency while minimizing data loss, as only individual attributes needed to be deleted rather than entire tuples. [8]

CHAPTER 7

Relations between the Scores

The described scores are for explanations for classification outcomes or to better perform database repairs. But are the explanations of the scores all equally good? And are their values all the same? In this chapter we ask ourselves whether there are relations between the described scores and what these relations look like. We will concentrate on the results from the field of machine learning, because on the one hand this field is already widely researched and therefore there are more results and on the other hand most of the results can be applied relatively easily to the problem of database repairs anyway.

We will also show comparisons that are almost self-evident or have already been discussed in individual chapters. Nevertheless, we will mention them here for the sake of completeness, in order to provide a holistic picture of the scores.

Furthermore, we will use the following trivial example to illustrate the relations with the calculation of the individual scores:

Example 7.0.1. A fire can only burn if the three ingredients oxygen, heat and fuel are always available. If one of the three is missing, the fire cannot continue to burn. From this, a trivial classifier $\mathcal{C}$ can be built, which for each of the eight possible combinations of the three binary features $oxygen/no\ oxygen$, $heat/no\ heat$ and $fuel/no\ fuel$ can tell whether a fire can burn in such an environment.

For our example, we will use the letters **O**, **H**, and **F** to abbreviate the feature names $oxygen$, $heat$, and $fuel$. We will further use the entity $\mathbf{e} = \{oxygen, no\ heat, fuel\}$ and calculate the different scores for the feature $heat$.

7.1 SHAP and Banzhaf Power Index

As described in Section 5.4.1, there is a trivial relationship between the SHAP score and the Banzhaf power index. This is because the latter consider not the permutations of

teams, but the combinations of teams. Consequently, the scores of the features naturally differ because the individual differences in the expected values are weighted differently. Furthermore, it can happen that not only the two scores are different, but one can even show differences in the ranking of the features. Nonetheless, SHAP and Banzhaf power index are very similar because they both have to compute the same expected values, which are only weighted differently. The calculation method also leads to the computational complexity of the scores, which is the same for both scores.

Example 7.1.1. To calculate the SHAP score for Example 7.0.1, all possible expected values must be calculated beforehand:

$$\begin{array}{llll} E[\mathcal{C}|\mathbf{e}_{\emptyset}] = \frac{7}{8} & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{O}\}}] = \frac{3}{4} & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{O},\mathbf{H}\}}] = 1 & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{O},\mathbf{H},\mathbf{F}\}}] = 1 \\ & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{H}\}}] = 1 & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{O},\mathbf{F}\}}] = \frac{1}{2} & \\ & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{F}\}}] = \frac{3}{4} & E[\mathcal{C}|\mathbf{e}_{\{\mathbf{H},\mathbf{F}\}}] = 1 & \end{array}$$

These expected values are then weighted as follows:

$$\begin{aligned} \mathrm{SHAP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) &= \frac{0!2!}{3!}\cdot\left(1-\frac{7}{8}\right) + \frac{1!1!}{3!}\cdot\left(1-\frac{3}{4}\right) + \frac{1!1!}{3!}\cdot\left(1-\frac{3}{4}\right) + \frac{2!0!}{3!}\cdot\left(1-\frac{1}{2}\right) \\ &= \frac{2}{6}\cdot\frac{1}{8} + \frac{1}{6}\cdot\frac{1}{4} + \frac{1}{6}\cdot\frac{1}{4} + \frac{2}{6}\cdot\frac{1}{2} \\ &= \frac{1}{24} + \frac{1}{24} + \frac{1}{24} + \frac{4}{24} = \frac{7}{24} \end{aligned}$$

The Banzhaf power index, on the other hand, uses exactly the same expected values, but weights them differently and thus produces a different result:

$$\begin{aligned} \mathrm{SHAP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) &= \frac{1}{4}\cdot\left(1-\frac{7}{8}\right) + \frac{1}{4}\cdot\left(1-\frac{3}{4}\right) + \frac{1}{4}\cdot\left(1-\frac{3}{4}\right) + \frac{1}{4}\cdot\left(1-\frac{1}{2}\right) \\ &= \frac{1}{4}\cdot\frac{1}{8} + \frac{1}{4}\cdot\frac{1}{4} + \frac{1}{4}\cdot\frac{1}{4} + \frac{1}{4}\cdot\frac{1}{2} \\ &= \frac{1}{32} + \frac{2}{32} + \frac{2}{32} + \frac{4}{8} = \frac{9}{32} \end{aligned}$$

So we can clearly see that the computational effort is the same, but the weighting is different. This results in two different scores, where the Banzhaf power index, unlike the SHAP score, does not have the properties described in Section 5.2 and is therefore less used in practice.

7.2 Banzhaf Power Index and Causal Effect

Since both the Banzhaf power index and the causal effect are used in the field of databases, it makes more sense to compare them this way instead of reformulating both definitions

for a classifier. As already mentioned in Section 5.4, both scores are equivalent. Due to the associativity of addition, Banzhaf power index can also be defined as follows:

$$\text{BPI}_{\mathcal{Q}}(D,\tau) = \frac{\sum_{T\subseteq D\setminus\{\tau\}} \mathcal{Q}(T\cup\{\tau\})}{2^{|D|-1}} - \frac{\sum_{T\subseteq D\setminus\{\tau\}} \mathcal{Q}(T)}{2^{|D|-1}}$$

The two terms here correspond exactly to the two expected values for the query $\mathcal{Q}$ on the probabilistic database as they appear in Definition 6.2.1. Unfortunately, the computational complexity is the same as for the Shapley values, so in practice the latter are used more often.

Example 7.2.1. Since the two scores are equivalent, the value for the causal effect $\text{CE}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H})$ for the running example is also $\frac{9}{32}$, like the $\text{BPI}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H})$ in Example 7.1.1.

7.3 SHAP and COUNTER

In [9], the SHAP score is defined in several steps, called *levels*. Each so-called level-based SHAP score, denoted by lb-SHAP, is only part of the total SHAP score, so that when all lb-SHAP scores are summed up, the total SHAP score is obtained. This is done by splitting the sum in Definition 5.3.2 into multiple sums that no longer iterate over all subsets, but only over those of a certain size, the level. Thus, intuitively, the levels represent the size of the teams in the cooperative game that underlie the SHAP score. The formal definition of the lb-SHAP score is as follows:

Definition 7.3.1 (LEVEL-BASED SHAP) [9]. Let $\mathcal{C}$ be a classifier and Pr be a data distribution, then the lb-SHAP score for level l, feature $f\in\mathcal{F}$ and entity $\mathbf{e}\in\mathcal{E}$ is:

$$\text{lb-SHAP}_{\mathcal{C},Pr}(\mathbf{e},f,l) = \sum_{F\subseteq\mathcal{F}\setminus\{f\},|F|=l} \frac{l!(|\mathcal{F}|-l-1)!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{F\cup\{f\}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_F])$$

Note, that the only difference to the SHAP score is the size of the set F.

If the level of the lb-SHAP score is set to $l=|\mathcal{F}|-1$, there is only one set for which $F\subseteq\mathcal{F}\setminus\{f\}$ holds. The relation between the SHAP score and the COUNTER score is therefore:

$$\begin{aligned}
\text{lb-SHAP}_{\mathcal{C},Pr}(\mathbf{e},f,l) &= \frac{l!(|\mathcal{F}|-l-1)!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus\{f\}}]) \\
&= \frac{l!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus\{f\}}]) \\
&= \frac{l!}{|\mathcal{F}|!}(\mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus\{f\}}]) \\
&= \frac{l!}{|\mathcal{F}|!}\text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e},f) \\
&= \frac{1}{|\mathcal{F}|}\text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e},f)
\end{aligned}$$

Here it can be seen that COUNTER score and SHAP score are quite different, because the SHAP score has not only one level, but $|\mathcal{F}|$ different levels. Also the complexity is very different, because with the COUNTER score only one level of the SHAP score can be calculated, but for the SHAP score all levels are needed to be computed to sum them up. [9]

Example 7.3.2. Consider again the running example of this chapter. Notice that only two of the expected values calculated in Example 7.1.1 are needed to compute the COUNTER score. This makes the calculation much easier, but also less accurate, since it does not include all the information. The COUNTER score is then calculated as follows:

$$\text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) = \mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\{O,F\}}] = 1 - \frac{1}{2} = \frac{1}{2}$$

Similarly, two expected values are sufficient to calculate the lb-SHAP score:

$$\text{lb-SHAP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H},2) = \frac{2!0!}{3!} \cdot \left(1 - \frac{1}{2}\right) = \frac{2}{6} \cdot \frac{1}{2} = \frac{1}{6}$$

The following equation can be used to verify that the values of the two scores are in the correct ratio to each other:

$$\text{lb-SHAP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H},2) = \frac{1}{|\mathcal{F}|} \cdot \text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$$

For the SHAP score, however, all levels of the lb-SHAP score must be calculated. This makes the calculation of the SHAP score, as shown in Example 7.1.1, very different from that of the COUNTER score.

7.4 RESP and Causal Effect

In Chapter 6, the RESP score and the causal effect are introduced. These were defined because the x-RESP score is too unintuitive. For a feature f the x-RESP score only counts the features whose values necessarily need to be changed to make the actual cause f a counterfactual cause, but not how likely the counterfactual cause also changes the outcome. This problem is solved by the RESP score and the causal effect, but in different ways. The RESP score uses a combination of the COUNTER and x-RESP scores. For the causal effect, a probabilistic database is required initially, through which the probabilities of the outcomes with and without the feature can be calculated. Due to their different calculation methods, there is no real relationship between the two scores, except that they both solve the problem of the x-RESP score.

Example 7.4.1. For the RESP score, it needs a contingency set. We start with the smallest possible set, the empty set. Since we already know from Example 7.3.2 that

$\text{COUNTER}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) > 0$, we also know that the feature *heat* must be a counterfactual cause for the entity $\mathbf{e}$. As a result, the contingency set $\emptyset$ is already sufficient to change the result with a single change of the feature *heat*. This results in the following RESP score:

$$\text{RESP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H},\emptyset) = \frac{\mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\{O,F\}}]}{1+|\emptyset|} = \frac{1-\frac{1}{2}}{1} = \frac{1}{2}$$

Since $\emptyset$ is the smallest contingency set, with $\text{RESP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H},\emptyset) > 0$ we have

$$\text{RESP}_{\mathcal{C},Pr}(\mathbf{e},\mathbf{H}) = \frac{1}{2}.$$

In this particular example, the RESP score also requires only two of the expected values calculated in Example 7.1.1, since the contingency set is $\emptyset$. This greatly simplifies the calculation, requiring far fewer expected values to be calculated than when calculating the causal effect, since the calculation can be stopped early. As we have shown in Example 7.2.1, not only the calculation method but also the results of the two scores are different, so there is no direct relation between them.

7.5 SHAP and RESP

As we have just seen, COUNTER score and SHAP score are already relatively dissimilar. Since the RESP score is only equal to the COUNTER score if the contingency set is 0, RESP score and SHAP score are even more different. Despite the general similarity in complexity, their computations are very different. The SHAP score originates from the Shapley values and the RESP score derives from causality. The different underlying principles then give rise to the different mathematical definition. [9]

What is exciting, however, is that both scores have their own advantages and disadvantages. The SHAP score is already very well researched. This has already allowed us to define problem classes where the computation works in polynomial time. An example of this is mentioned in Section 5.3.1, where we proved that the SHAP score can be computed in polynomial time for all features of a binary classifier with a fully factorized distribution of the data. A disadvantage of the SHAP score is that the computation requires a large number of conditional expected values and these cannot be properly estimated from a sample. In contrast, the RESP score does not need to calculate every possible contingency set in many cases. Often a small contingency set or even an empty contingency set is sufficient to compute the RESP score. Especially in real cases, there are rarely large contingency sets, or they make little sense there. [9]

Example 7.5.1. The advantages and disadvantages just described are also reflected in the calculations. In Example 7.1.1 and Example 7.4.1 it can be seen that the calculations are very different, bring different results and in special edge cases are even significantly easier than the other. Therefore, the underlying definitions do not invoke any relations between the two scores.

CHAPTER 8

Explanations for Sets of Features

With all the scores for single features it seems straightforward to extend these to arbitrary coalitions. Note that it is not just the explanation of a single feature, but the explanation of the combined set of features. The question we ask in this chapter is: how much contribution does an arbitrary set of features S have to the outcome of a classifier? For this we redefine SHAP and COUNTER to form a generalization of the relation in Section 7.3. We call the two new definitions SHAP$^+$ and COUNTER$^+$ respectively and will introduce them first in the following two sections before we then redefine the relation.

8.1 Definition of SHAP$^+$

We start with the definition of the SHAP$^+$ score. This is the extension of the SHAP score to sets of features and is defined as follows:

Definition 8.1.1 (SHAP$^+$). Let $\mathcal{C}$ be a classifier and Pr be a data distribution, then the SHAP$^+$ score of a set of features $S \subseteq \mathcal{F}$ of an entity $\mathbf{e} \in \mathcal{E}$ is:

$$\text{SHAP}^+{}_{\mathcal{C},Pr}(\mathbf{e}, S) = \sum_{F \subseteq \mathcal{F} \setminus S} \frac{|F|!|S|!(|\mathcal{F}| - |F| - |S|)!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{F \cup S}] - E_{Pr}[\mathcal{C}|\mathbf{e}_F])$$

Note that the definition is very similar to that of SHAP, except that the numerator is multiplied by $|S|!$. Recall that in the numerator of Definition 5.3.2 we multiplied the number of permutations of the set F by the number of permutations of the set $|\mathcal{F}| - |F| - 1$ to enumerate all permutations where feature $f \in \mathcal{F}$ is at position $|F| + 1$. Now, instead of a single feature $f \in \mathcal{F}$, we have a set of features $S \subseteq \mathcal{F}$ that can also take different orders in a permutation of $\mathcal{F}$. In the definition of SHAP$^+$ the factor $|S|!$ was added to include these permutations.

It is easy to show that unfortunately not all of the properties described in Section 5.2 are valid for the SHAP$^+$ score. Nevertheless, this score is highly relevant because it

calculates not only the contribution of individual features, but also the contribution of combinations of features. In practice, there are cases where multiple feature values can be changed, and instead of using the top features with the highest SHAP scores, the SHAP$^+$ score can be used to calculate the best combination of features to change the result of the classifier.

8.2 Definition of COUNTER$^+$

The second definition we want to present here is the COUNTER$^+$ score. This uses the idea of the COUNTER score, where a single feature, a so-called counterfactual cause, makes the difference between the real and the desired classifier outcome. If the probability is high that a change in the value of the feature of a certain entity e will also change the classifier outcome, then the feature has a high COUNTER score for e. To find explanations for sets of features, such a counterfactual cause must consist of the change of the value of several features, whereby it can be computed whether a change of the classifier outcome is likely. The COUNTER$^+$ score is therefore defined as follows:

Definition 8.2.1 (COUNTER$^+$). Let $\mathcal{C}$ be a classifier and Pr be a data distribution, then the COUNTER$^+$ score for features $S \subseteq \mathcal{F}$ and entity $\mathbf{e} \in \mathcal{E}$ is:

$$\text{COUNTER}^+_{\mathcal{C},Pr}(\mathbf{e}, S) = \mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus S}]$$

This definition is similar to the underlying Definition 6.1.1 of the COUNTER score, except that instead of a single feature f, we now use a set of features S. However, this set S is different from the contingency set of the RESP score. Recall that the contingency set is a set of features that must be changed before calculating the probability of obtaining the desired classifier outcome by changing the value of a single feature f. However, the set for the COUNTER$^+$ score is used to calculate the probability of obtaining the desired classifier outcome by changing all feature values in the set S.

8.3 Generalization of the Relation between SHAP and COUNTER

In Section 7.3 we saw that by using the lb-SHAP score we can establish a relationship between SHAP and COUNTER. Now we try to use the generalizations of the mentioned scores to create a new relation between SHAP$^+$ and COUNTER$^+$.

For this also the SHAP$^+$ score can be level-based, in which we can fix the size of the subsets in the sum:

$$\text{lb-SHAP}^+_{\mathcal{C},Pr}(\mathbf{e}, S, l) = \sum_{F \subseteq \mathcal{F}\setminus S, |F|=l} \frac{l!|S|!(|\mathcal{F}| - l - |S|)!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{F\cup S}] - E_{Pr}[\mathcal{C}|\mathbf{e}_F])$$

Note that the same factor had to be added to this definition as to Definition 8.1.1, since the order of the features of the set S is relevant here as well. Otherwise, this definition is similar to Definition 7.3.1 from Section 7.3.

If the level l is restricted to the form $l = |\mathcal{F}| - |S|$, where S is an arbitrary subset of the features $\mathcal{F}$. Note that due to the restricted form of the level, the conditions in the sum would only apply to a single set of features. Then the relation between SHAP$^+$ and COUNTER$^+$ follows immediately:

$$\begin{aligned}
\text{lb-SHAP}^+_{\mathcal{C},Pr}(\mathbf{e}, S, l) &= \frac{l!|S|!(|\mathcal{F}| - l - |S|)!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus S}]) \\
&= \frac{l!|S|!}{|\mathcal{F}|!}(E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}}] - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus S}]) \\
&= \frac{l!|S|!}{|\mathcal{F}|!}(\mathcal{C}(\mathbf{e}) - E_{Pr}[\mathcal{C}|\mathbf{e}_{\mathcal{F}\setminus S}]) \\
&= \frac{l!|S|!}{|\mathcal{F}|!}\text{COUNTER}^+_{\mathcal{C},Pr}(\mathbf{e}, S) \\
&= \binom{|\mathcal{F}|}{|S|}^{-1}\text{COUNTER}^+_{\mathcal{C},Pr}(\mathbf{e}, S)
\end{aligned}$$

This relation is the generalization of the relation in Section 7.3. Thus, as already shown in Section 7.3, it is clear that both the computation and therefore also the complexity of the SHAP$^+$ score is much more difficult than that of COUNTER$^+$ score, since in the relation only a single level of the SHAP$^+$ score has to be calculated. Nevertheless, it is interesting to note that there is a relationship between the Shapley value based score and the causality based score even with set of features.

A final word on the RESP score with sets of features: Of course, it is possible to extend the RESP score for sets of features and define a RESP$^+$ score similar to what we did at the beginning of this chapter. However, we have already seen in the last chapter in Section 7.5 that there are no meaningful relations between SHAP and RESP. During our research we came to the same results with sets of features, since responsibility is a completely different concept then the Shapley values. Therefore, we came to the conclusion that two scores can only be compared by their empirically results. We discuss this comparison at the end of the next chapter, after we have presented our own implementation of these scores.

CHAPTER 9

Implementation and Experimental Evaluation

In order to work with the scores, to develop them further or to carry out tests, implementations and their evaluations are necessary. We will divide this chapter into two main parts. First, we will present the existing implementation by Bertossi in [7], which allows us to calculate at least the x-RESP score, and second, we will present our own implementation for all the scores mentioned so far. For a better understanding, we will also show examples and finally present and discuss an evaluation, possible extensions and improvements of the implementation.

9.1 Counterfactual Intervention Programs

First, let's look at Bertossi's implementation in [7]. There he introduces the notion of *Counterfactual Intervention Programss (CIPs)*. CIPs are ASPs that can compute counterfactual explanations for a classifier through multiple interventions. Since entities are needed for the input to the classifier, they are represented as predicates of the form $E(id, f_1, \ldots, f_n, state)$:

- The first argument is the id of the entity to find all related entities after the program starts. When dealing exclusively with individual entities, the inclusion of id can be omitted entirely or substituted with a constant value. Therefore the constant e is used in the examples.
- The next n arguments describe the entity in more detail. The variables $f_1, \ldots, f_n$ represent the n properties of the entity.
- The last argument specifies the status of the entity. The following values are allowed $\{o, do, tr, s\}$, which are explained in the general description of CIPs.

From this, the initial entity $\vec{f} = \langle f_1, \ldots, f_n \rangle$ gives rise to a predicate $E(e, \vec{f}, o)$, where the state o marks the "original" entity. In his examples, Bertossi assumes that the undesired output is 1 and the desired output is 0. To emphasise that this is not necessary, we replace the two constants by `bad` and `good` respectively. Since Bertossi assumes that the classifier assigns the value 1 to the initial entity, we assume that the initial entity receives the value `bad`. [7]

A classifier is also required for a CIP, but the concrete implementation of the classifier is irrelevant for the CIP. Here are possible ways to implement it, which we will use and discuss later:

- For each entity $\vec{f} = \langle f_1, \ldots, f_n \rangle$, facts of the form $\mathcal{C}(\vec{f}, prediction)$ can be defined, where the variable *prediction* can take the values `bad` or `good` and thus determines the classification of the entity $\vec{f}$.
- The facts just described can also be defined by rules and calculated dynamically.
- An external program can be used that is called and executed for each classification. In this way, for example, existing classifiers in other programming languages can be used.

We will show and discuss examples for all three types in this chapter. But first, let us look at the general structure of a CIPs:

P1 First, the facts of the program are defined. These consist of the domains of the features $Dom(i, c)$ with $c \in Dom_i$ and $E(e, \vec{f}, o)$ with $\vec{f}$ as features of the initial entity.

P2 Then all entities are put into the state "transition", abbreviated tr. It is only in this state that small changes are made in an attempt to find an entity that achieves the desired result, `good`. This is why we have the following formulas:

$$E(e, \vec{f}, tr) \rightarrow E(e, \vec{f}, o).$$
$$E(e, \vec{f}, tr) \rightarrow E(e, \vec{f}, do).$$

P3 Any resulting entity that does not produce the desired result is marked with "do" to initiate a new change. To do this, the non-deterministic "choice-operator" chooses a new value for a change for each entity whose prediction yields `bad`. Of course, this choice must be restricted to the domain of the respective feature, and at the same time the entity must actually change as a result of the new value. This leads to the following rules:

$$E(e, f_1'.f_2, \ldots, f_n, do) \vee \cdots \vee E(e, f_1.f_2, \ldots, f_n', do) \rightarrow E(e, \vec{f}, do), \mathcal{C}(\vec{f}, bad),$$
$$Dom(f_1'), \ldots, Dom(f_n'),$$
$$f_1 \neq f_1', f_n \neq f_n',$$
$$choice(\vec{f}, f_1'), \ldots, choice(\vec{f}, f_n').$$

where $choice(\vec{f}, f_i')$ means to choose a new value f_i' for the feature i and replace it in the entity $\vec{f}$. Here $choice(\vec{f}, f_i)$ is only meant to be comprehensible, but can be replaced by the predicate $chosen_i(\vec{f}, c)$, which can be defined by classical rules:

$$Chosen_i(\vec{f}, c) \rightarrow E(e, \vec{f}, tr), \mathcal{C}(\vec{f}, bad), Dom(c), c \neq f_i, notDiffChoice_i(\vec{f}, c).$$
$$DiffChoice_i(\vec{f}, c) \rightarrow Chosen_i(\vec{f}, c'), c' \neq c.$$

P4 The program stops when our result matches the desired result, `good`. To do this, we introduce a rule that creates an entity with status s, for "stop", at the given time:

$$E(e, \vec{f}, s) \rightarrow E(e, \vec{f}, do), \mathcal{C}(\vec{f}, good).$$

P5 Of course, we don't want the program to revert to the original entity by making changes, so we need to add the following rule to prevent this:

$$\rightarrow E(e, \vec{f}, do), E(e, \vec{f}, o).$$

P6 Finally, all that remains is to form explanations with the available facts. All we need to do is to check which features have changed from the original entity and create a new predicate $Expl(e, i, f_i)$. i stands for the feature and f_i for the original value that had to be changed to get the result `good`. The resulting rule for $i = 1, \ldots, n$ is

$$Expl(e, i, f_i) \rightarrow E(e, \vec{f}, o), E(e, \vec{f'}, s), f_i \neq f_i'.$$

The goal of a CIP is stable models. Because of the choice operator and the disjoint rule P3, it is possible that there are multiple stable models for a CIP. Each of the stable models contains modified versions of the initial entity and also one that is annotated with the status s and thus is a counterfactual version of the initial entity. If the program is unsatisfiable, so there are no stable models, there is no way to modify the initial version so that an entity with status s can emerge, and therefore no explanations for the result. [7]

The following example should make the functionality of CIPs clearer. We are using the *DLV* system [28]. We will also use the *DLV-Complex* system [11] for numeric aggregation and set operators, and finally show how to use external programs as classifiers with the newer version *DLV2* [2, 1], which follows the ASP Core 2 standard [12]. [7]

Example 9.1.1. Consider Example 3.1.1. We now present the CIP for the decision tree in Figure 3.2. Compared to Bertossi's [7], we have made some minor changes to make it easier to understand and consistent with our definitions.

```
1 % facts
2 dom(outlook,sunny). dom(outlook,cloudy). dom(outlook,rain).
```

```
3  dom(humidity,high). dom(humidity,normal).
4  dom(wind,strong). dom(wind,weak).
5
6  % original entity
7  E(e,sunny,normal,weak,o).
8
9  % specification of the classifier
10 cls(sunny,high,strong,no). cls(sunny,high,weak,no).
11 cls(sunny,normal,strong,yes). cls(sunny,normal,weak,yes).
12 cls(cloudy,high,strong,yes). cls(cloudy,high,weak,yes).
13 cls(cloudy,normal,strong,yes). cls(cloudy,normal,weak,yes).
14 cls(rain,high,strong,no). cls(rain,high,weak,yes).
15 cls(rain,normal,strong,no). cls(rain,normal,weak,yes).
16
17
18 % transition rules
19 E(e,O,H,W,tr) :- E(e,O,H,W,o).
20 E(e,O,H,W,tr) :- E(e,O,H,W,do).
21
22 % counterfactual rule
23 E(e,Op,H,W,do) v E(e,O,Hp,W,do) v E(e,O,H,Wp,do) :-
24                 E(e,O,H,W,tr), cls(O,H,W,yes),
25                 dom(outlook,Op), dom(humidity,Hp), dom(wind,Wp),
26                 O!=Op, H!=Hp, W!= Wp,
27                 chosen(outlook,O,H,W,Op),
28                 chosen(humidity,O,H,W,Hp),
29                 chosen(wind,O,H,W,Wp).
30
31 % definitions of chosen operators
32 chosen(F,O,H,W,U) :- E(e,O,H,W,tr), cls(O,H,W,yes),
33                 dom(F,U), U!=O, not diffchoice(F,O,H,W,U).
34 diffchoice(F,O,H,W,U) :- chosen(F,O,H,W,Up), U!=Up, dom(F,U).
35
36 % not going back to initial entity
37 :- E(e,O,H,W,do), E(e,O,H,W,o).
38
39 % stop when label has been changed
40 E(e,O,H,W,s) :- E(e,O,H,W,do), cls(O,H,W,no).
41
42 % collecting changed values for each feature
43 expl(e,outlook,O) :- E(e,O,H,W,o), E(e,Op,Hp,Wp,s), O!=Op.
44 expl(e,humidity,H) :- E(e,O,H,W,o), E(e,Op,Hp,Wp,s), H!=Hp.
45 expl(e,wind,W) :- E(e,O,H,W,o), E(e,Op,Hp,Wp,s), W!=Wp.
46
47 % auxiliary predicate
48 % to avoid unsafe negation in the constraint below
49 entAux(e) :- E(e,O,H,W,s).
50 % discard models where label does not change
51 :- E(e,O,H,W,o), not entAux(e).
```

```
52
53 % computing the inverse of x-Resp
54 invResp(e,M) :- #count {F: expl(e,F,_)}=M, #int(M).
```

Algorithm 9.1: CIP for the decision tree in Figure 3.2. [7]

Lines 2-4 first define the domains for *outlook*, *humidity* and *wind*. Then the initial entity is defined in line 6. The classifier is specified explicitly in lines 8-13. Note that the classifier can also be defined implicitly by rules as in Algorithm 9.2.

```
1 cls(O,H,W,yes) :- O=sunny, H=normal, dom(wind,W).
2 cls(O,H,W,yes) :- O=cloudy, dom(humidity,H), dom(wind,W).
3 cls(O,H,W,yes) :- O=rain, W=weak, dom(humidity,H).
4 cls(O,H,W,no)  :- dom(outlook,O), dom(humidity,H), dom(wind,W),
5                                                  not cls(O,H,W,yes).
```

Algorithm 9.2: Additional rules for the decision tree in Figure 3.2. [7]

Another way to specify the classifier would be to represent it by an external program, as was done in Algorithm 9.3 in Python.

```
1 def classifier(O,H,W):
2     if (O == "sunny") and (H == "normal"):
3         return "yes"
4     if (O == "overcast"):
5         return "yes"
6     if (O == "rain") and (W == "weak"):
7         return "yes"
8     else:
9         return "no"
```

Algorithm 9.3: External Python program for the decision tree in Figure 3.2. [7]

All that needs to be done is to compute the predicate *cls* at the beginning by calling the function in the Python program with $\&classifier(O, H, W; L)$. This can be done with DLV2 as long as the path to the Python Program 9.3 is passed as the second parameter in the program call. [7]

Once the facts and classifier are defined, the rules in lines 19 and 20 bring the initial entity $E(e, sunny, normal, weak, o)$ into its transition state $E(e, sunny, normal, weak, tr)$. Thus, the rules in lines 32 and 34 become active and produce all possible changes to the individual feature values. The rule in line 23 then creates counterfactual versions of the initial entity. Important here is line 37, so that the calculation does not start over, and the rule in line 40, so that the program stops as soon as a counterfactual version is found.

Finally, the explanations for each feature are collected in lines 43-45. Counting the explanations, which are the features that have changed, gives the inverse x-RESP score. The rule in line 52 creates a score for each stable model. The predicate *invResp* has been introduced to make it more convenient to read the score. [7]

The only problem with this CIP is that the quality of the stable models as explanations varies widely. Let's take a closer look at the two resulting stable models:

```
{E(e,sunny,normal,weak,o),
    cls(sunny,normal,strong,1),cls(sunny,normal,weak,1),
    cls(overcast,high,strong,1),cls(overcast,high,weak,1),
    cls(rain,high,weak,1),cls(overcast,normal,weak,1),
    cls(rain,normal,weak,1),cls(overcast,normal,strong,1),
    cls(sunny,high,strong,0),cls(sunny,high,weak,0),
    cls(rain,high,strong,0),cls(rain,normal,strong,0),

    E(e,sunny,high,weak,do),E(e,sunny,high,weak,tr),
    E(e,sunny,high,weak,s),

    expl(e,humidity,normal),invResp(e,1)
}

{E(e,sunny,normal,weak,o),
    cls(sunny,normal,strong,1),...,cls(rain,normal,strong,0),

    E(e,rain,normal,strong,do),E(e,rain,normal,strong,tr),
    E(e,rain,normal,strong,s),
    expl(e,outlook,sunny),expl(e,wind,weak),invResp(e,2)
}
```

Both stable models contain the facts described above and the classifier. In addition, they both have one counterfactual entity each, which is marked with the status s and would be labeled *no* by the classifier. Note that each entity that is labeled *no* by the classifier would have its own model if we did not stop the computation as soon as such an entity is reached. Thus, changing the feature *wind* of the counterfactual version of the first stable model would create the entity $E(e, sunny, high, strong)$, which would also receive the label *no*. The same is true for the second stable model, the feature *humidity*, and the resulting counterfactual version of the original entity $E(e, rain, high, strong)$. However, the two counterfactual versions of the initial entity differ mainly in the number of modified features. The first stable model requires only one change to the initial entity, while the second requires two. This value can also be read from the predicate *invResp*. As we know, a single explanation of the classifier's result for the initial entity is more meaningful and thus more important than two explanations. Therefore, the first stable model is "better" than the second. The stable models are thus s-explanations. However, we are mainly interested in the c-explanation. In order to obtain only the c-explanation, we only need to add weak constraints that minimize the changes. In other words, we have to minimize the number of explanations. These weak constraints, unlike strong constraints, can be violated, but the violation must be kept as small as possible. The following rules accomplish this:

$$\rightsquigarrow E(e, \vec{f}, o), E(e, \vec{f'}, s), x_i \neq x'_i.$$

So for our program only these DLV rules would be missing:

```
% weak constraints for minimizing number of changes:
:~ E(E,O,H,W,o), E(E,Op,Hp,Wp,s), O != Op.
:~ E(E,O,H,W,o), E(E,Op,Hp,Wp,s), H != Hp.
:~ E(E,O,H,W,o), E(E,Op,Hp,Wp,s), W != Wp.
```

This leaves only the stable models with the minimum number of changed features needed to compute the x-RESP score. [7]

Note that this program is completely generic except for the initial entity. Therefore, it is possible to compute the inverse x-RESP score for any and even multiple entities. As an example, entities can be read from a separate file and subsequently provided to the program as input. [7]

Through Example 9.1.1 we have seen how to use CIPs to compute the x-RESP score of one or more entities. A natural question is whether this also works with the other scores described in this thesis. During our research we noticed a new probabilistic extension called *plingo* [20] for *clingo*, another system besides DLV for solving logic programs. This one was promising because, at least we thought, it also allowed different distributions of the data for the classifier. Unfortunately, the systems for solving logic problems are not yet advanced enough to continue computing with the probabilistic values. Therefore, we decided to implement the scores in Python. This implementation will be explained and discussed in detail in the following section, before we perform an evaluation with real data.

9.2 Python Implementation

As shown in the last section, there are ways to calculate the x-RESP score using logic programs. To compute other scores, we decided to create a framework in Python that allows us to use real datasets and classifiers. In the following we will describe the different parts of the framework and try to make them understandable.

Since data is used throughout the entire computation, we need to take a closer look at data preprocessing at the beginning. Before the data can be used, it has to be prepared. For example, some classifiers require binary features, some feature values need to be scaled, or records with missing information need to be deleted or modified. All of this must be done before the classifier can be trained. We also decided to keep the operation as simple as possible, and to store these transformation steps so that they can be used to modify any entity in the same way later, rather than having to calculate them by hand for each feature. For the storage and transformation we created a class `Preparer`, which can be completely customized by a configuration file. The data is read in CSV format and transformed into the `DataFrame` datatype of the Python package *pandas* at the beginning of the program. This has an advantage and a disadvantage. On the one hand, the data can be used quickly at any time, but on the other hand, the memory of the hardware is a limiting factor, since the data is completely in memory.

Once the data is read and processed, a classifier can be trained. In Section 9.1 we have seen different ways to define such a classifier for CIPs. Since we are using the Python programming language, which is widely used in the field of machine learning, there are almost no limits for the programmer to define the classifier. Therefore, we have created a highly parameterized Python framework that allows to train a classifier with a configuration file and to save the model with little effort. In our tests we mainly used simple classifiers like the `RandomForestClassifier` or the `GaussianNB` from the *scikit-learn* Python package, since our focus was on computing scores rather than good classifiers. However, by *Grid Search* the parameters of the mentioned classifiers, which means a repeated division of training and test data and the corresponding training, we were able to create quite accurate classifiers. It was especially important for us not to make the classifier too sensitive to our few datasets, in order to avoid so-called overfitting, which means tailoring the classifier to the given data. Otherwise, the classifier would not be as accurate for new datasets. In addition, the framework stores the model of the classifier so that we do not have to restart training for each score calculation.

The core of our score calculation implementation is the Python class `Explainer` with the following properties

- the `predict` function of the classifier
- the preprocessing steps to make an entity "classifier-ready"
- the domains of the features
- the function to distribute the entities

The `Explainer` class also contains two functions, namely `partial_entity` and `expected_prediction`. The former creates a `DataFrame` of a partial entity according to Definition 5.3.1. The latter computes the expected value of the predictions of a `DataFrame` of entities according to the stored distribution function.

For the score calculation, we created additional functions. Let's start with the trivial implementation of the COUNTER score:

```
1 def counter(self, entity, feature):
2     return self.expected_prediction(entity, set(entity.index)-{
      feature})
```

Algorithm 9.4: Python implementation of the COUNTER score

The first parameter of the function is `self`, which is an instance of the `explainer` class and thus specifies the setting. This parameter is present in all functions because we need the information stored in the `Explainer` class anyway. For this reason we will ignore it in the following function descriptions. The other parameters are `entity` and `feature`. `entity` is the entity we want to analyze and `feature` is the feature we want to compute the COUNTER value for. The expected value of the partial entity is

returned, where the second parameter of the `expected_prediction` function specifies the features to fix. In other words, it calculates the expected value of the entities where all feature values remain the same except for the value of the feature `feature`.

Almost self-explanatory is the very similar implementation of the COUNTER$^+$ score:

```
def counter_plus(self, entity, features):
    return self.expected_prediction(entity, set(entity.index)-set(
        features))
```

Algorithm 9.5: Python implementation of the COUNTER$^+$ score

Here, only the `feature` has been replaced by a set of features `features`. This changes the value for all features in the set `features` instead of just a single feature.

Probably the most complicated implementation is the x-RESP score:

```
def x_resp(self, entity, feature, max_size=None, get_counter=True):
    entity_prediction = self.predict(self.preprocess(entity.to_frame
        ().transpose()))[0]

    candidates = list(set(entity.index)-{feature})
    best_counter = entity_prediction
    max_size = len(candidates) if max_size is None else max_size-1
    for size in range(max_size+1):
        for contingency_set in combinations(candidates, size):

            entities = self.partial_entity(entity, set(entity.index)-
                set(contingency_set))
            bad_entities = entities[self.predict(self.preprocess(
                entities)) == entity_prediction]
            counters = bad_entities.apply(lambda pe: self.counter(pe,
                feature), axis=1)

            best_counter = max(best_counter, *counters, key=lambda c:
                abs(c-entity_prediction))
        if best_counter != entity_prediction:
            return (1/(1+size), best_counter) if get_counter else
                1/(1+size)
    return (0, 0) if get_counter else 0
```

Algorithm 9.6: Python implementation of the x-RESP score

In addition to the parameters already explained, there are two more optional parameters. The parameter `max_size` can be used to set the maximum size of the contingency set. One significant advantage of the RESP score, as discussed in Section 7.5, is the ability to terminate the score calculation early. This is possible due to the initial consideration of small contingency sets, as larger contingency sets tend to have diminished expressive power. To not have an early termination condition here, leave `max_size` to `None`, otherwise set the parameter to the maximum size of the contingency set that still makes

sense. The last parameter is `get_counter`. This parameter determines whether the best counter for the contingency set is also returned or just the contingency set. The function itself then tries to find the smallest possible contingency set by cleverly testing values and returns the x-RESP score of the entity `entity` and the feature `feature`. Since the COUNTER score function is present and frequently executed in this implementation, we can see that the x-RESP score is derived from the COUNTER score.

To combine the positive features of the COUNTER score and the x-RESP score, we have the RESP score. Once again, it is evident in the implementation that both the COUNTER score and the x-RESP score are used within the calculation of the RESP score:

```
1 def resp(self, entity, feature):
2     x_resp_score, best_counter = self.x_resp(entity, feature,
          get_counter=True)
3     return x_resp_score * best_counter
```

Algorithm 9.7: Python implementation of the RESP score

The parameters of this function are the same as for the COUNTER score and are used to calculate the x-RESP score including the best COUNTER score. The two results are then simply multiplied to return the RESP score.

For the SHAP score, we created our own Python class `COOP_Game`. This is capable of calculating Shapley scores. For the SHAP score, we simply compute the Shapley values for a given wealth function. Since Python is also a functional programming language, we could simply pass the appropriate function as a parameter. The implementation for the Shapley scores looks like this

```
1  class COOP_Game:
2      def __init__(self, game_function):
3          self.GAME_FUNCTION = game_function
4
5      def shapley(self, all_players, player):
6          shapley_value = 0
7          for size in range(len(all_players)):
8              coefficient = (factorial(size)*factorial(len(all_players)
                   -size-1)) / (factorial(len(all_players)))
9              for team in combinations(set(all_players.index)-set(
                   player), size):
10                 shapley_value += coefficient * (self.GAME_FUNCTION(
                       all_players, set(player).union(team)) - self.
                       GAME_FUNCTION(all_players, team))
11
12         return shapley_value
```

Algorithm 9.8: Python implementation of the COOP_Game class

The `COOP_Game` class stores the wealth function because it is necessary for further calculations. To compute Shapley values as in Example 5.1.2, the function `shapley`

needs the list of all players `all_players` and the one player `player` for which the Shapley value is to be computed. Then the Shapley values of the teams and their coefficients are calculated and multiplied. The sum calculated in line 10 is then the resulting Shapley score that is returned.

As previously mentioned, calculating the SHAP score simply requires the creation of a `COOP_Game` using the wealth function `expected_prediction` from the `Explainer` class. The resulting Shapley scores will be equivalent to the SHAP score. The creation of the `COOP_Game` object and the function call were implemented as follows:

```
1 def shap(self, entity, feature):
2     return COOP_Game(self.expected_prediction).shapley(entity, {
      feature})
```

Algorithm 9.9: Python implementation of the SHAP score

Like the COUNTER$^+$ score, the SHAP$^+$ score is very similar to the SHAP score. Again, a `COOP_Game` is created with the wealth function `expected_prediction`. This time, however, the function is called with a set of features:

```
1 def shap_plus(self, entity, features):
2     return COOP_Game(self.expected_prediction).shapley(entity, set(
      features))
```

Algorithm 9.10: Python Implementation of the SHAP$^+$ score

9.3 Evaluation

To test the framework, we used real data. In principle, the content and the statement of the data is not important for a test. Nevertheless we tried to find data that are similar to the examples in this thesis. To be able to use the continuous features, we discretized them to get categorical features where one can change feature values in a finite domain. In addition, we encoded all categorical data into multiple binary features using `OneHotEncoder` from the Python package `sklearn.preprocessing` to prevent the classifier from creating unwanted relationships between the values of individual features and thus achieve higher accuracy. The resulting domains can be found in the parenthesis next to the features.

- *Blood Transfusion Service Center Data Set (blood)* [43]: A relatively small dataset that can be used to predict whether people will donate blood. It contains 748 records of blood donors, each with 4 features and a binary class for classification:
  - `Recency`: Months since last donation (`<3, 3-4, 5-11, 12-16, >16`)
  - `Frequency`: Total number of blood donations (`<3, 3-4, 5-7, >7`)

  - `Monetary`: Total amount of blood donated in c.c.
    `(<501, 501-1000, 1001-1750, >1750)`
  - `Time`: Months since first donation
    `(<15, 15-23, 24-35, 36-57, >57)`
  - `Class`: Binary feature representing whether the donor was a blood donor in March 2007
    `('yes', 'no')`

  For this dataset, we used the `GaussianNB` classifier from the Python package `sklearn.naive_bayes` and set the parameter

  - `var_smoothing: 1.484`,

  for an accuracy of $0.7\overline{3}$.

- *Statlog (German Credit Data) Data Set (creditworthiness)* [26]: This is the largest dataset we have tested. It can be used to determine a person's credit score. It contains 1000 records with a total of 20 features. The features include the person's gender, education level, and whether the person rents or owns their home. The resulting binary class indicates whether the person is creditworthy (`good`) or not (`bad`). We discretized the following features:

  - `Age`: Age of person
    `(<28, 28-33, 34-42, >42)`
  - `Credit amount`: Amount of the credit
    `(<1367, 1367-2320, 2321-3973, >3973)`
  - `Duration`: Duration of the credit
    `(<13, 13-24, >24)`

  For this dataset, we used the `RandomForestClassifier` classifier from the Python package `sklearn.ensemble` and called it with the parameters

  - `n_estimators: 50`
  - `criterion': entropy`
  - `min_samples_split: 0.012`

  for an accuracy of 0.755.

- *German Credit Risk (creditworthiness-small)* [27]: This dataset is the same as the previous one and also contains all 1000 records, but only 9 features. It was generated from the previous dataset by omitting some features and better describing others. We use it mainly because it contains fewer features and thus reduces the number of possible value combinations.
  For this dataset, we again used the `GaussianNB` classifier from the Python package `sklearn.naive_bayes`, but this time we specified the parameter

- `var_smoothing: 1`

for a precision of 0.725.

To compute the scores, we selected multiple entities for each dataset that received a negative classification result and tried to determine which parameters were most likely to need to be changed to achieve a positive result. To do this, we selected the following entities:

- blood: Since the 4 features can take 5, 4, 4 and 5 different values, this gives 400 possible entities. We deleted those that did not give a negative result at the beginning. We were left with 393 entities for which we then calculated scores for all features.
- creditworthiness: Since this dataset contains a lot of possible entities due to the 20 features and up to 10 possible values per feature, we decided to take only those that actually occur in the dataset and give a negative result. From the remaining 216 entities, we then used the 66 for which there is also an equivalent entity for the creditworthiness-small dataset, which returned a negative result with the associated classifier. This makes the two datasets easy to compare as we used the same entities for both.
- creditworthiness-small: As just described, for this dataset we used the 66 entities that occur in the datasets and returned a negative result for both the classifier of the creditworthiness dataset and the creditworthiness-small dataset.

Note that the last two entity sets describe the same people, so only the size of the dataset differs, not the local case under study.

In addition, we used the uniform distribution, the fully-factorized distribution and the experimental distribution as parameters to compute both the scores of the features for the selected entities and to investigate how long the computation takes for the different distributions.

The evaluation was carried out on a standard laptop equipped with an Intel® Core™ i7-12700H processor and 16 GB of RAM. This setup was complemented by an Arch Linux operating system running Kernel version 6.4.4-arch1-1 (64-bit). Opting for a common laptop configuration was a deliberate choice to ensure the applicability of our results across a wide range of real-world scenarios, without being influenced by specialized hardware accelerators or excessive computational resources. This decision aligns with our commitment to prioritizing simplicity and accessibility in our hardware choices, thus promoting a collaborative and sustainable framework for advancements in the field. The entire calculation of all the scores, took over 9 hours.

9.3.1 Scores for single Features

In Figure 9.1, 9.2 and 9.3, we have shown the computed scores for the features from the blood, creditworthiness-small and creditworthiness datasets, respectively. We used boxplots to get a sense of which features are relevant to a change in classifier outcome. Note that in Figure 9.3, due to the size of the dataset, we can only display the features that are similar to those in the creditworthiness-small dataset.

If we look at the results for the blood dataset in Figure 9.1, we see that for any feature, in any distribution, the COUNTER value is almost always 0. This means that there are very few entities with a single feature for which there is another value that would produce a counterfactual version of the entity. It also means that calculating the other causality-based scores is not trivial, as it usually requires a contingency set to be found first. We will see this again later in Figure 9.4: the calculation of the RESP score is no faster than that of the SHAP score for the blood dataset. The same can be seen with the x-RESP score: There are only a few outliers with a x-RESP score of 1, which means that no contingency set is needed. Furthermore, the median of the x-RESP scores for the features `Frequency` and `Monetary` is $0.\overline{3}$, which means that half of all entities need at most two additional features to turn a negative result of the classifier into a positive one. However, the range is relatively large and there are some entities that require three additional features or whose classification result cannot be changed at all. This is different from the feature `Recency` because here, apart from a few outliers, all entities need two or three features in the contingency set. On the other hand, besides the fact that a single feature in the contingency set is not sufficient, this means that virtually any entity can be modified so that the feature `Recency` can be decisive for the classification result of the modified entity. The low x-RESP values for the feature `Time` show that this feature can only be decisive for the result of the classifier by changing the values of almost all other features. Furthermore, it can be clearly seen that the RESP score is always smaller or equal to the x-RESP score found with the contingency set, since the RESP score becomes smaller and smaller as the contingency sets become larger and larger.

The SHAP score, on the other hand, is quite different and always near 0. As we saw in Section 5.2, the SHAP scores can be summed up to the difference $\mathcal{G}(\mathcal{P}) - \mathcal{G}(\emptyset)$. For example, the result of the classifier for the entity $\langle 2, 11, 2750, 23\rangle$ of the blood dataset is 1.0 and the expected value of the classifier with uniform distribution is 0.9825. Thus, the difference of 0.0175 is equal to the sum of all SHAP scores for this configuration, except for rounding errors resulting from the Python calculation:

$$0.014375 - 0.093125 - 0.093125 + 0.189375 = 0.0175$$

Of course, other distributions will have different expected values and thus different results, but they can still be summed up to this difference. This is why the SHAP values are so close to 0.

In the creditworthiness-small dataset examined in Figure 9.2, more COUNTER scores are not equal to 0 than in the blood dataset. Therefore, there are many x-RESP scores

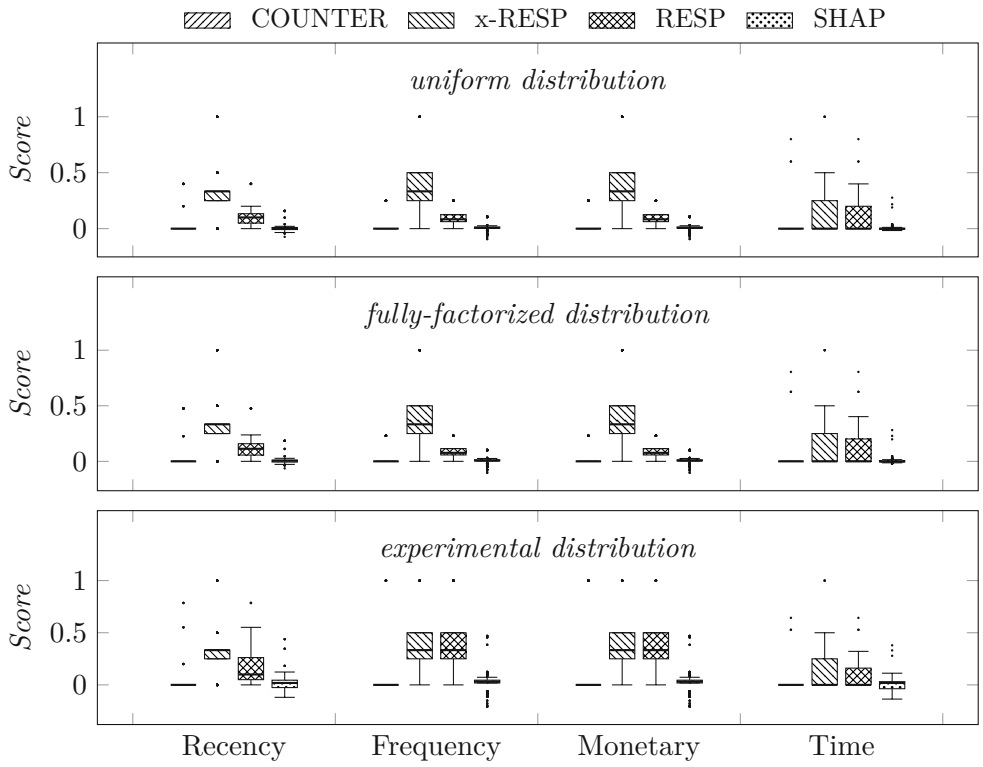

Figure 9.1: Scores for the entities of the blood dataset.

of 1, which results from an empty contingency set. This makes the COUNTER scores and the RESP scores similar. The features `Saving accounts`, `Checking account` and `Duration` are even counterfactual causes for almost all entities. The same features emerge when the SHAP scores of each feature are considered. These features have on average the highest SHAP scores and therefore contribute the most to the classifier's result. However, by stopping the search for the smallest possible contingency sets, it is possible to compute the causality-based scores faster than the SHAP scores, thus providing similar results with less effort.

Due to the same data origin, the results for the creditworthiness dataset are very similar to the results for the creditworthiness-small dataset. This can also be seen in Figure 9.3. On the one hand, we see many high RESP scores for the features on the right side of the figure, which indicates a large influence on the classifier outcome. On the other hand, we see some features with COUNTER score 0 for almost all entities on the left side of the figure. However, for the latter, usually only a second feature needs to be changed to make them counterfactual causes, since the x-RESP score is 0.5 for almost every entity.

In general, the results of the uniform distributions and the fully-factorized distributions of all datasets are very similar. However, the experimental distribution is often very different from the other two. This difference is due to the fact that the dataset has only

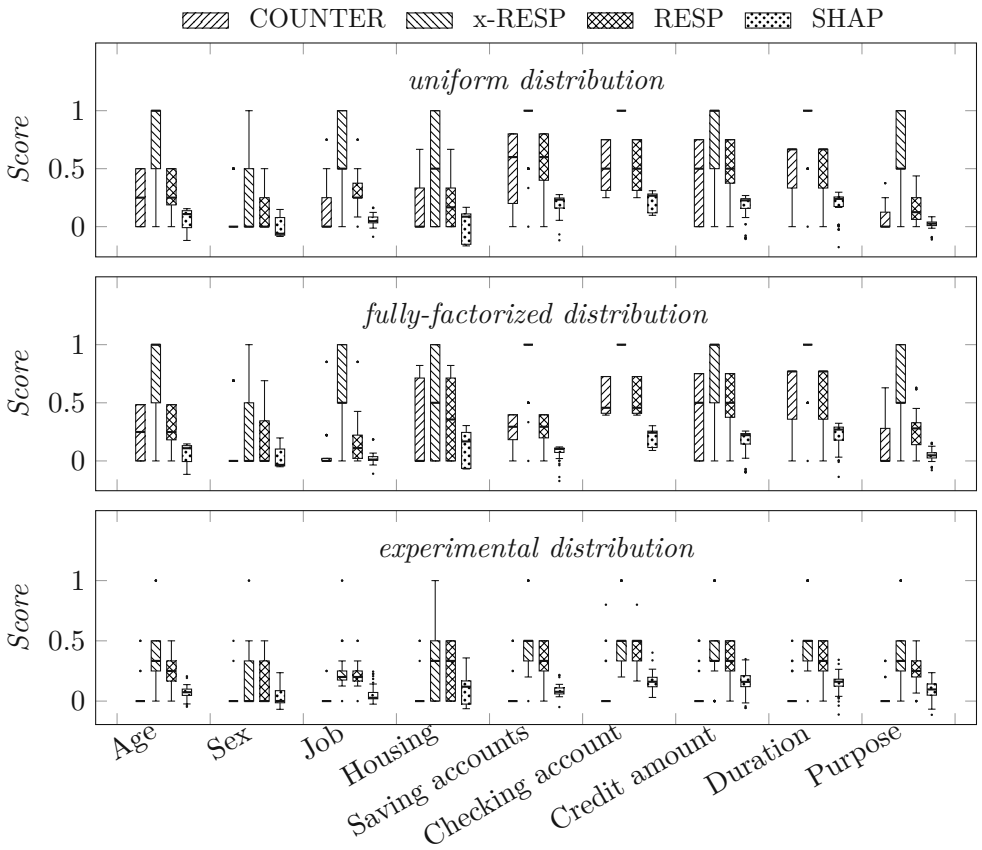

Figure 9.2: Scores for the entities of the creditworthiness-small dataset.

1000 records, which reduces the probability of any modified entities occurring at all. This bias does not occur in the first two distributions because no entity has a probability of 0.

The reason why the dataset is too small is also the reason why we could not compute the scores for the experimental distribution of creditworthiness at all. It would require too many contingency sets, which would require computing too many expected values. The same is true for calculating the SHAP score. For the 20 features of the creditworthiness dataset, each with two to ten different possible values, there are individual expected values for which more than $20 \cdot 10^9$ possible entities would have to be classified and then weighted according to the distribution used. All these entities need to be created and stored, so the 16 GB of memory used would not be enough. For the same reasons, the values for the x-RESP score and RESP score for the creditworthiness record cannot be computed.

Figure 9.4 illustrates again that for small instances the SHAP score can be computed quickly, but for larger instances the RESP score can be computed faster due to the few expected values that need to be computed, since in practice often small contingency sets

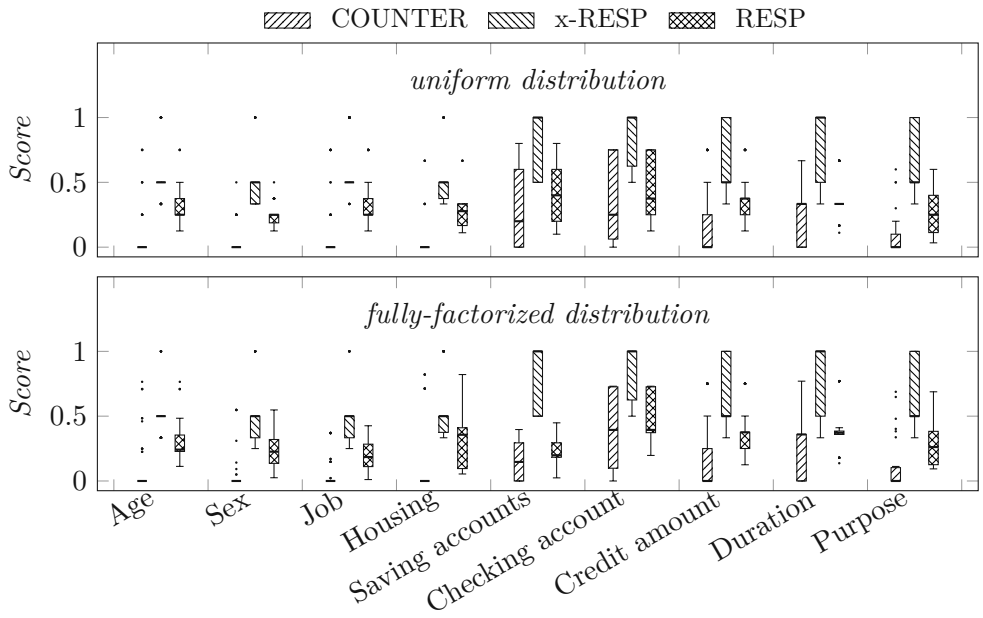

Figure 9.3: Scores for the entities of the creditworthiness dataset.

are sufficient to change the classifier outcome of an entity.

On the other hand, the expected values computed for a feature during the computation of the first SHAP score can be used to compute the score of further features faster. All possible expected values must be computed at least once, but this can be done in advance. Afterwards, the values just need to be saved and can then be used. However, this is not noticeable in Figure 9.4, as it shows the average runtime for computing the scores of single entities. Storing expected values with the RESP score is of little or no use, however, since expected values of different entities must be computed in most cases. Since it is not clear in advance which entities these are, too many expected values would have to be computed and stored, so it is not worthwhile to compute the values in advance.

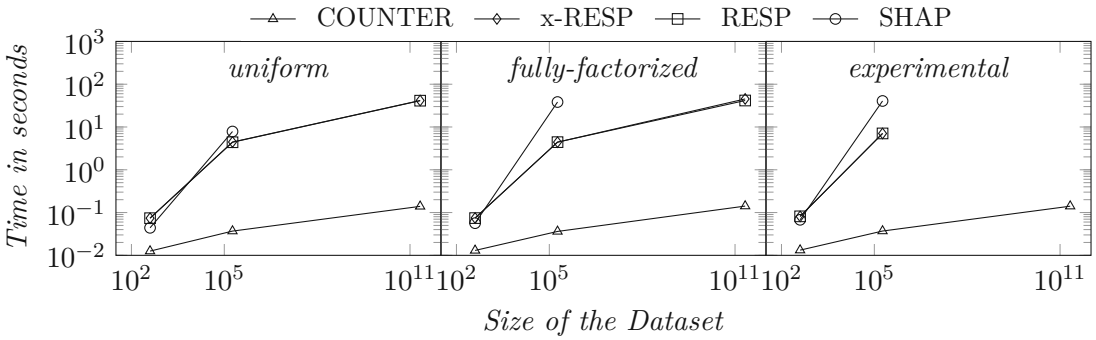

Figure 9.4: Average computation time of the scores.

9.3.2 Scores for Sets of Features

Due to the high complexity and huge computational effort of calculating the scores for all feature sets, we could not calculate them for the creditworthiness dataset. For the other two datasets, however, we again used the entities described earlier and focused primarily on whether they rank the feature combinations in the same way, since the values do not have any relations per se.

However, there were significant differences in the correlations between the two datasets. In the table 9.1 we have listed the average correlations of the rankings of of the COUNTER$^+$ score and the SHAP$^+$ score. It quickly becomes clear that the rankings for the blood dataset are very different and have a correlation of about 0.27 in the best case. The rankings of the creditworthiness-small dataset, on the other hand, are almost identical with a correlation of over 0.93. This large difference is due to the fact that many COUNTER$^+$ scores cannot be ranked properly at all. For the majority of entities examined in the blood dataset, many feature combinations have a COUNTER$^+$ score of 0. This does not define which feature combination has more influence on the classifier outcome. Since this does not happen as often in the creditworthiness-small dataset, the results are much better and also show that the two scores for sets of features tend to be highly correlated in their feature combination ranking.

Furthermore, it is noticeable that for both scores there is a tendency that feature combinations with features that were ranked high by the COUNTER score or the SHAP score were also ranked high in the ranking of the COUNTER$^+$ score and the SHAP$^+$ score, and vice versa.

Dataset \ Distribution	uniform	fully-factorized	experimental
blood	0.15029187	0.17459961	0.270176622
creditworthiness-small	0.94185869	0.94254958	0.93906126

Table 9.1: Correlation of COUNTER$^+$ score and SHAP$^+$ score.

In terms of computation time, the COUNTER$^+$ score and the SHAP$^+$ score are almost identical, as can be seen in Figure 9.5. This is again due to the fact that they have to compute the same expected values. However, they do so at different times. The COUNTER$^+$ score calculates a single expected value for each feature combination, while the SHAP$^+$ requires each expected value calculated by the COUNTER$^+$ score for each individual feature combination. This results in the calculation times of an exemplary entity shown in Figure 9.5. The COUNTER$^+$ score computes the expected values continuously, with the latter combinations containing more entities and taking longer to compute a single expected value. The SHAP$^+$ score, on the other hand, computes all the expected values at the first feature combination and stores them so that they do not have to be computed again for the other combinations. The storage is important because otherwise each call would take the same time as the first one. Again, this is a problem

when there is not enough memory for the results. This storage is not necessary with the COUNTER$^+$ score.

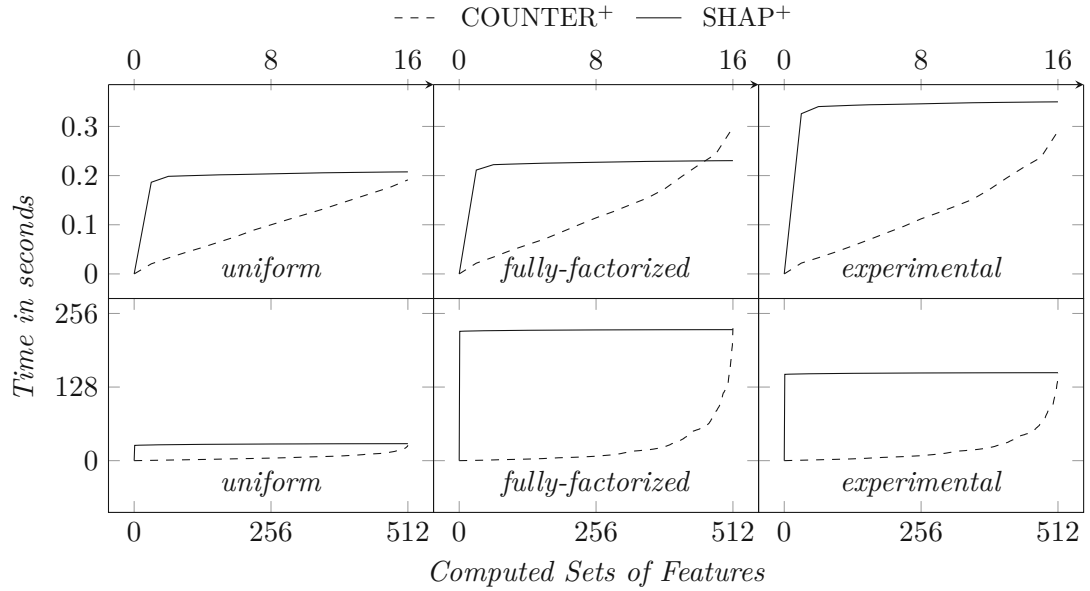

Figure 9.5: Computation time of the scores for sets of features.

9.4 Future Extensions

Of course, the described implementation of our score framework can be improved. For further research in the future, it will be important to be able to compute the scores faster and to create even more possibilities for customization.

Again, let us start with the data. The choice of data and its pre-processing strongly influences the result. Since we were only doing tests for the computation, we could discretize the data arbitrarily and even change it. This is not possible for real applications. Future research could focus on minimizing the need to prepare the data and perhaps even allow continuous features. Choosing the right classifier is also important. More options to use different data, preprocessing and classifiers would help to get results faster with less effort and fewer changes to the data.

Also, the choice of distributions and scores could be expanded to create more comparisons. It is important to note that very few distributions are discussed in the literature, and a scientific focus should be placed on this theoretical area.

Finally, it should be noted that our implementation is deliberately very naïve in order to make the definitions as understandable as possible. Also, the choice of data structures and the Python programming language itself, while understandable, are not as efficient as possible. Of course, faster algorithms can be implemented in more special cases, as we

have shown for example in Section 5.3.1, but this would miss our goal of explaining the scores and their relationships.

CHAPTER 10

Conclusion

In conclusion, the field of explainable data management and eXplainable Artificial Intelligence (XAI) is a rapidly advancing area of research, gaining increasing importance with a strong emphasis on transparency, fairness and explainability in machine learning and data management systems. The demand for explanations is growing constantly, underscoring the critical need to delve deeper into the notions of explanation and interpretation.

Throughout this thesis, we have extensively explored various facets of explanation in the realms of machine learning and databases, providing a comprehensive overview of the fundamental principles. Additionally, we have examined the role of causality within the context of explanations, with a particular focus on the concept of responsibility. Our analysis has encompassed score-based explanations utilizing Shapley scores in both machine learning and databases, shedding light on their intricacies and complexities.

Furthermore, we have ventured into the integration of causality and responsibility into predictive explanations, specifically exploring the realm of counterfactual interventions. Here, our primary focus has centered around the COUNTER, x-RESP, and RESP scores proposed by Bertossi et al. in [9]. We have conducted an extensive comparative analysis of these different scores while also introducing two novel scores derived from the COUNTER and SHAP scores, namely COUNTER$^+$ score and SHAP$^+$ score. To illustrate the practical implementation of these conceptual approaches, we have demonstrated and elucidated counterfactual intervention programs developed by Bertossi in [7]. Recognizing the challenges associated with computing scores beyond the x-RESP score, we have taken the initiative to develop our own Python framework, providing a comprehensive description of the entire explanation workflow. This framework, complemented by an evaluation of the proposed methods, offers a practical tool for both researchers and practitioners. It is our belief that this contribution will facilitate further advancements in the field of explanation.

Nevertheless, the pursuit of comprehensive and effective metrics for evaluating the quality of explanations remains a pressing research challenge. By identifying and defining the desired properties of explanation scores, researchers can pave the way for the development of robust evaluation measures. Additionally, the continued evolution of the framework, along with novel approaches incorporating probabilistic extensions for logic programs, holds the potential to enhance the analysis of explanations in complex and uncertain scenarios.

As we look toward the future of XAI, it is imperative that we confront these challenges head-on and strive to establish comprehensive frameworks for generating high-quality explanations. The concepts, methods, and tools presented in this thesis offer a solid foundation upon which future research can build. Their practical applicability across diverse domains further underscores their value and potential impact.

In summary, this thesis has contributed to the understanding and application of explanations in the fields of machine learning and databases. The insights gained from this research provide valuable resources for explanations of predictive models and can be leveraged in various application areas. The developed Python framework not only demonstrates its utility in practical scenarios but also serves as a catalyst for further advancements in the field. With continued exploration, refinement, and collaborative efforts, we can chart a course towards a future where high-quality explanations are an integral part of intelligent systems, fostering trust, understanding, and responsible use of AI technology.

CHAPTER 11

Future Work

This thesis has dealt with score-based explanations and tried to look at them from different perspectives and different research areas. In this chapter we will now look at possible extensions of the topic. We will ask ourselves which aspects have not yet been sufficiently researched and which new methods could be used in the future to advance research in this area. Furthermore, this chapter will serve to uncover open questions and challenges that have not yet been resolved or should be considered in further proceedings.

11.1 I cannot change my Age

Central to the notion of counterfactual interventions is the practice of changing input parameters to induce a change in outcomes, as seen in scenarios such as classification tasks. But some things cannot be changed, or not without considerable effort. An example of this is the age of a person. Consider again Example 1.0.1 from the introduction: if the bank employee would use one of the scores defined in the last chapters to make an explanation, the best way to get a loan might be to change the age of the bank customer. This is hardly possible (or only possible with a lot of time and even then only in one direction) and therefore certainly not a reasonable explanation for not getting the credit. Another example is the citizenship of a bank client. This is also impossible or very difficult to change.

So there are explanations that simply do not translate to concrete actions to change the outcome. Finding these in the appropriate domain is sometimes not easy, but must be the first step to be able to calculate useful scores. Excluding single features from the calculation of the scores could be a way to get no explanations for them. In general, if the feature to be excluded reaches the highest score, it is not enough to simply take the feature with the next lower score. Excluding individual features can alter the scores of all features, which are highly dependent on each other in the presented score calculations. The RESP score, for example, uses a contingency set. If a feature that cannot be changed

is in the contingency set, it is not sufficient to simply remove it from the set. The original contingency set would not be minimal if that were possible.

So features that are not allowed to be changed are something that has to be considered in future research. It is necessary to find solutions that can handle this and to define new scores with which a subsequent change of the set of changeable features is possible.

11.2 Probability Distributions

Another problem mentioned in [9] concerns the probability distribution. If the expected values are not precise, no score is of any use, because the resulting explanations are not accurate enough to be used. The choice of the probability distribution is therefore essential. In this thesis we have looked at the uniform, the fully-factorized and the experimental distribution. If one has more information about the field of application, one can achieve great advances. It depends on how likely it is that certain combinations of feature values occur. If these are ignored, a lot of information is lost, which could help to calculate the expected values more exactly and faster. Of course, one could also calculate the expected values exactly, if one has the data for it, but this helps only little for new entities and is mostly very time-consuming. [7, 9]

Example 11.2.1. Suppose we have data with two features A and B with their domains $dom(A) = \{a_1, a_2, \ldots, a_n\}$ and $dom(B) = \{b_1, b_2, \ldots, b_n\}$. Without looking at the data specifically, one might assume a fully factorized distribution and perhaps overlook the fact that the two features are not independent, but always occur in a pair: $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$. The expected value and thus the probability for a single entity would be $\frac{1}{n^2}$, although the single entities have a probability of $\frac{1}{n}$, that is, far more likely to occur.

Often minor dependencies and relationships are not easy to find, but they can greatly affect the result. A solution to keep the information about the frequencies of the combinations of the feature values is to define them explicitly. We will look at this in more detail in Section 11.2.1. [7]

11.2.1 Custom Probability Distributions

Wherever standardized distributions are no longer sufficient or too little data is available for experimental distributions, these can be customized. This has the purpose to capture more of the correlations and logical relationships between the features. To calculate the probability of an entity e in a population $\mathcal{E}$ we first assume a probability distribution Pr. For Pr we introduce so-called *constraints*, which are very similar to the denial constraints in databases. [7]

These use the set of all binary features $\mathcal{F}$ and are of the form:

$$\chi : \neg \left(\bigwedge_{F \in \mathcal{F}_1} \bar{F} \wedge \bigwedge_{F' \in \mathcal{F}_2} \bar{F'} \right)$$

where $\mathcal{F}_1 \cup \mathcal{F}_2 \subseteq \mathcal{F}$, $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$ and $\bar{F}$ and $\bar{F'}$ mean that features F and F' take values 1 and 0, respectively. Intuitively, these describe the absence of certain pairs of feature values in a population. To use the constraint χ in the custom probability distribution Pr_χ, we define the event $E(\chi) = \{e \in \mathcal{E} \mid e \models \chi\}$. Here $e \models \chi$ means that e satisfies the constraint χ. [7]

Given the initial probability space $\langle \mathcal{E}, Pr \rangle$, we can now define the customized probability space $\langle \mathcal{E}, Pr_\chi \rangle$ by

$$Pr_\chi(E) = Pr(E|E(\chi)) = \frac{Pr(E \cap E(\chi))}{Pr(E(\chi))}$$

where $E \in (E)$. Thus Pr is well-defined as long as $E(\chi) \neq \emptyset$. The probability of the violation set of χ is:

$$Pr_\chi(\mathcal{E} \setminus E(\chi)) = Pr(\mathcal{E} \setminus E(\chi)|E(\chi)) = \frac{Pr(\emptyset)}{Pr(E(\chi))} = 0$$

Furthermore, the definition for finite conjunctions of constraints $\bigwedge \theta$ and arbitrary propositional formulas f, which can be evaluated for single entities, can be extended by using $Pr_{\bigwedge \theta}(E)$ or $Pr_f(E)$ respectively. The resulting customized version of the probability distribution can be used in all described scores to calculate the expected values as accurately as possible. [7]

Example 11.2.2. Let us assume that the two features in Example 11.2.1 represent two binary properties of a person:

- *Adult*: Is the person an adult
- *highIncome*: If the person earns a lot

Most likely, the two properties *Adult* and *highIncome* have a correlation. Using the full-factorized distribution as in Example 11.2.1 would lead to inaccurate results. It is rare to have children who have a very high income. Thus, value pairs would be included in the calculation, which do not occur at all or only rarely. The constraint $\neg(\neg Adult \wedge highIncome)$ prevents these combinations and thus leads to a more accurate expected value of the entities. Furthermore, other constraints can be added to further improve the result. [7]

In any case, the two improvements described in this chapter should be considered and perhaps even compared in future work. Expected values, which were created without including the context, lead to wrong results and make a correct calculation and thus the interpretation of the scores impossible.

List of Figures

List of Tables

List of Algorithms

Acronyms

ASP answer-set program. 7, 11, 12, 61

BCQ Boolean Conjunctive Query. 7, 16, 42

blood Blood Transfusion Service Center Data Set. 71, 73–75, 78

CIP Counterfactual Intervention Programs. 61–63, 65–68, 87

CQ Conjunctive Query. 7, 42

creditworthiness Statlog (German Credit Data) Data Set. 72–78

creditworthiness-small German Credit Risk. 72–76, 78

XAI eXplainable Artificial Intelligence. 81, 82

Bibliography

[1] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, and F. Ricca. Evaluation of disjunctive programs in WASP. In M. Balduccini, Y. Lierler, and S. Woltran, editors, *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings*, volume 11481 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2019.

[2] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, and J. Zangari. The ASP system DLV2. In M. Balduccini and T. Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 215–221. Springer, 2017.

[3] M. Arenas, P. Barceló, L. E. Bertossi, and M. Monet. The tractability of shap-scores over deterministic and decomposable boolean circuits. *CoRR*, abs/2007.14045, 2020.

[4] M. Arenas, P. Barceló, L. E. Bertossi, and M. Monet. On the complexity of shap-score-based explanations: Tractability via knowledge compilation and non-approximability results. *CoRR*, abs/2104.08015, 2021.

[5] R. Bagheri. Introduction to shap values and their application in machine learning. `https://towardsdatascience.com/introduction-to-shap-values-and-their-application-in-machine-learning-8003718e6827`, Mar 2022. [Online; accessed 03-April-2022].

[6] L. Bertossi and G. Reyes. Answer-set programs for reasoning about counterfactual interventions and responsibility scores for classification. *CoRR*, abs/2107.10159, 2021.

[7] L. E. Bertossi. Declarative approaches to counterfactual explanations for classification. *CoRR*, abs/2011.07423, 2020.

[8] L. E. Bertossi. Score-based explanations in data management and machine learning: An answer-set programming approach to counterfactual analysis. *CoRR*, abs/2106.10562, 2021.

[9] L. E. Bertossi, J. Li, M. Schleich, D. Suciu, and Z. Vagena. Causality-based explanation of classification outcomes. In S. Schelter, S. Whang, and J. Stoyanovich, editors, *Proceedings of the Fourth Workshop on Data Management for End-To-End Machine Learning, In conjunction with the 2020 ACM SIGMOD/PODS Conference, DEEM@SIGMOD 2020, Portland, OR, USA, June 14, 2020*, pages 6:1–6:10. ACM, 2020.

[10] L. E. Bertossi and B. Salimi. From causes for database queries to repairs and model-based diagnosis and back. *Theory Comput. Syst.*, 61(1):191–232, 2017.

[11] F. Calimeri, S. Cozza, G. Ianni, and N. Leone. An ASP system with functions, lists, and sets. In E. Erdem, F. Lin, and T. Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, pages 483–489. Springer, 2009.

[12] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, and T. Schaub. Asp-core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309, 2020.

[13] H. Chockler and J. Y. Halpern. Responsibility and blame: a structural-model approach. *CoRR*, cs.AI/0312038, 2003.

[14] H. Chockler, J. Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *CoRR*, cs.LO/0312036, 2003.

[15] G. V. den Broeck, A. Lykov, M. Schleich, and D. Suciu. On the tractability of SHAP explanations. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6505–6513. AAAI Press, 2021.

[16] T. Eiter and T. Lukasiewicz. Causes and explanations in the structural-model approach : Tractable cases. In A. Darwiche and N. Friedman, editors, *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pages 146–153. Morgan Kaufmann, 2002.

[17] T. Eiter and T. Lukasiewicz. Complexity results for structure-based causality. *Artif. Intell.*, 142(1):53–89, 2002.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[19] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

[20] S. Hahn, T. Janhunen, R. Kaminski, J. Romero, N. Rühling, and T. Schaub. Plingo: A system for probabilistic reasoning in clingo based on LP $^{\text{mln}}$. In G. Governatori and A. Turhan, editors, *Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR 2022, Berlin, Germany, September 26-28, 2022, Proceedings*, volume 13752 of *Lecture Notes in Computer Science*, pages 54–62. Springer, 2022.

[21] J. Y. Halpern. Defaults and normality in causal structures. In G. Brewka and J. Lang, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pages 198–208. AAAI Press, 2008.

[22] J. Y. Halpern. A modification of the halpern-pearl definition of causality. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3022–3033. AAAI Press, 2015.

[23] J. Y. Halpern and C. Hitchcock. Graded causation and defaults. *CoRR*, abs/1309.1226, 2013.

[24] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach, part I: causes. *CoRR*, cs.AI/0011012, 2000.

[25] C. Hitchcock and J. Knobe. Cause and norm. *Journal of Philosophy*, 106(11):587–612, 2009.

[26] H. Hofmann. Statlog (german credit data) data set. `https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)`, Nov 1994. [Online; accessed 16-May-2023].

[27] U. M. Learning. German credit risk. `https://www.kaggle.com/datasets/uciml/german-credit`, Sep 2019. [Online; accessed 16-May-2023].

[28] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[29] D. Lewis. Causation. *Journal of Philosophy*, 70(17):556–567, Oct 1973.

[30] E. Livshits, L. E. Bertossi, B. Kimelfeld, and M. Sebag. The shapley value of tuples in query answering. *CoRR*, abs/1904.08679, 2019.

[31] E. Livshits, L. E. Bertossi, B. Kimelfeld, and M. Sebag. Query games in databases. *SIGMOD Rec.*, 50(1):78–85, 2021.

[32] S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017.

[33] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.

[34] P. Menzies. *Causation in context*, pages 191–223. Clarendon Press, 2007.

[35] T. M. Mitchell. *Machine learning, International Edition.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[36] C. H. Papadimitriou. On the complexity of unique solutions. *J. ACM*, 31(2):392–400, 1984.

[37] S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010.

[38] B. Salimi and L. E. Bertossi. From causes for database queries to repairs and model-based diagnosis and back. *CoRR*, abs/1412.4311, 2014.

[39] B. Salimi, L. E. Bertossi, D. Suciu, and G. V. den Broeck. Quantifying causal effects on query answering in databases. *CoRR*, abs/1603.02705, 2016.

[40] L. S. Shapley. A value for n-person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.

[41] M. Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997.

[42] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[43] I.-C. Yeh. Blood transfusion service center data set. `https://www.openml.org/search?type=data&status=active&sort=runs&qualities.NumberOfFeatures=lte_10&id=1464`, Oct 2008. [Online; accessed 16-May-2023].

[44] F. Zhu. A new halpern-pearl definition of actual causality by appealing to the default world. *Axiomathes*, pages 1–20, Jan 2022.