Data Mining and Knowledge Discovery
https://doi.org/10.1007/s10618-024-01016-z

A comparative study of methods for estimating model-agnostic Shapley value explanations

Lars Henry Berge Olsen[1,2] · **Ingrid Kristine Glad**[1] · **Martin Jullum**[3] · **Kjersti Aas**[3]

Received: 16 May 2023 / Accepted: 21 February 2024

Abstract
Shapley values originated in cooperative game theory but are extensively used today as a model-agnostic explanation framework to explain predictions made by complex machine learning models in the industry and academia. There are several algorithmic approaches for computing different versions of Shapley value explanations. Here, we consider Shapley values incorporating feature dependencies, referred to as conditional Shapley values, for predictive models fitted to tabular data. Estimating precise conditional Shapley values is difficult as they require the estimation of non-trivial conditional expectations. In this article, we develop new methods, extend earlier proposed approaches, and systematize the new refined and existing methods into different method classes for comparison and evaluation. The method classes use either Monte Carlo integration or regression to model the conditional expectations. We conduct extensive simulation studies to evaluate how precisely the different method classes estimate the conditional expectations, and thereby the conditional Shapley values, for different setups. We also apply the methods to several real-world data experiments and provide recommendations for when to use the different method classes and approaches. Roughly speaking, we recommend using parametric methods when we can specify the

Responsible editor: Panagiotis Papapetrou

✉ Lars Henry Berge Olsen
lholsen@math.uio.no

Ingrid Kristine Glad
glad@math.uio.no

Martin Jullum
jullum@nr.no

Kjersti Aas
kjersti@nr.no

[1] Department of Mathematics, University of Oslo, Oslo, Norway

[2] The Alan Turing Institute, London, UK

[3] Norwegian Computing Center, Oslo, Norway

data distribution almost correctly, as they generally produce the most accurate Shapley value explanations. When the distribution is unknown, both generative methods and regression models with a similar form as the underlying predictive model are good and stable options. Regression-based methods are often slow to train but quickly produce the Shapley value explanations once trained. The vice versa is true for Monte Carlo-based methods, making the different methods appropriate in different practical situations.

Keywords Explainable artificial intelligence · Shapley values · Model-agnostic explanation · Prediction explanation · Feature dependence · Feature importance

1 Introduction

Complex machine learning (ML) models are extensively applied to solve supervised learning problems in many different fields and settings: cancer prognosis (Kourou et al. 2015), credit scoring (Kvamme et al. 2018), impact sensitivity of energetic crystals (Lansford et al. 2022), and money laundering detection (Jullum et al. 2020). The ML methods are often very complex, containing thousands, millions, or even billions of tuneable model parameters. Thus, understanding the complete underlying decision-making process of the ML algorithms is infeasible (for us humans). The use of ML methods is based on them having the potential to generate more accurate predictions than established statistical models, but this may come at the expense of model interpretability, as discussed by Johansson et al. (2011), Guo et al. (2019), Luo et al. (2019). Rudin (2019) conjectures that equally accurate but interpretable models exist across domains even though they might be hard to find.

The lack of understanding of how the input features of the ML model influence the model's output is a major drawback. Hence, to remedy the absence of interpretation, the fields of explainable artificial intelligence (XAI) and interpretable machine learning (IML) have become active research fields in recent years (Adadi and Berrada 2018; Molnar 2022; Covert et al. 2021). Various explanation frameworks have been developed to extract hidden knowledge about the underlying data structure captured by the black-box model, making the model's decision-making process more transparent. Model transparency is essential for, e.g., medical researchers who apply an intricate ML model to obtain well-performing predictions but who simultaneously also aim to discover important risk factors. The *Right to Explanation* legislation in the European Union's General Data Protection Regulation (GDPR) has also been a driving factor (European Commission 2016).

One of the most commonly used explanation frameworks in XAI is *Shapely values*, which is an explanation methodology with a strong mathematical foundation and unique theoretical properties from cooperative game theory (Shapley 1953). Shapley values are most commonly used as a *model-agnostic* explanation framework for individual predictions, that is, for *local explanations*. Model-agnostic means that Shapley values do not rely on model internals and can be used to compare and explain any ML model trained on the same supervised learning problem. Local explanation means that Shapley values explain the local model behavior for a specific observation and not the global model behavior across all observations. The methodology has also been used to provide *global explanations*, see, e.g., Owen (2014), Covert et al. (2020), Frye et al. (2021), Giudici and Raffinetti (2021). See Molnar (2022) for an overview and detailed introduction to other explanation frameworks.

Shapley values originated in cooperative game theory but have been reintroduced as a model explanation framework by Strumbelj and Kononenko (2010), Strumbelj and Kononenko (2014), Lundberg and Lee (2017). Originally, Shapley values described a possible solution concept of how to fairly allocate a game's payout among the players based on their contribution to the overall cooperation/payout. The solution concept is based on several desirable axioms, for which the Shapley values are the unique solution. When applying Shapley values as an explanation framework, we treat the features as the players, the predictive model as the game, and the corresponding prediction as the payout.

There are several ways to define the game, which yields different types of Shapley values. For local explanations, the two main types are *marginal* and *conditional* Shapley values[1], and there is an ongoing debate about when to use them (Chen et al. 2020; Kumar et al. 2020; Chen et al. 2022). Briefly stated, the marginal version ignores dependencies between the features, while the conditional version incorporates them. Thus, a disadvantage of the conditional Shapley values, compared to the marginal counterpart, is that they require the estimation/modeling of non-trivial conditional expectations. However, they are robust against adversarial attacks, which the marginal Shapley values are not (Blesch et al. 2023). Throughout this article, we refer to conditional Shapley values when discussing Shapley values if not otherwise specified.

There is a vast amount of literature on different approaches for estimating Shapley values (Strumbelj et al. 2009; Lundberg and Lee 2017; Lundberg et al. 2018; Redelmeier et al. 2020; Williamson and Feng 2020; Aas et al. 2021a, b; Frye et al. 2021; Covert et al. 2021; Olsen et al. 2022). These methods can be grouped into different method classes based on their characteristics, that is, if they (implicitly) assume feature independence or use empirical estimates, parametric assumptions, generative methods, and/or regression models; see Fig. 1. To the best of our knowledge, there exists no thorough and methodological comparison between all the method classes and approaches. Chen et al. (2022, Sect. 6) states that "[conditional Shapley values] constitutes an important future research direction that would benefit from new methods or systematic evaluations of existing approaches".

In this article, we both investigate existing methods, introduce several new approaches, and conduct extensive simulation studies starting from a very simple set-up with an interpretable model as a sanity check and gradually increase the complexity of the predictive model. We also investigate the effects the data distribution, with varying levels of dependence, and the training sample size have on the estimation of the conditional expectations using the different methods. Finally, we also conduct experiments on real-world data sets from the UCI Machine Learning Repository. In the numerical simulation studies, the parametric methods, which correctly (or nearly correctly) assume the data distribution, generate the most accurate Shapley values. However, if the data distribution is unknown, such as for most real-world data sets,

[1] They are also called *interventional* and *observational* Shapley values, respectively.

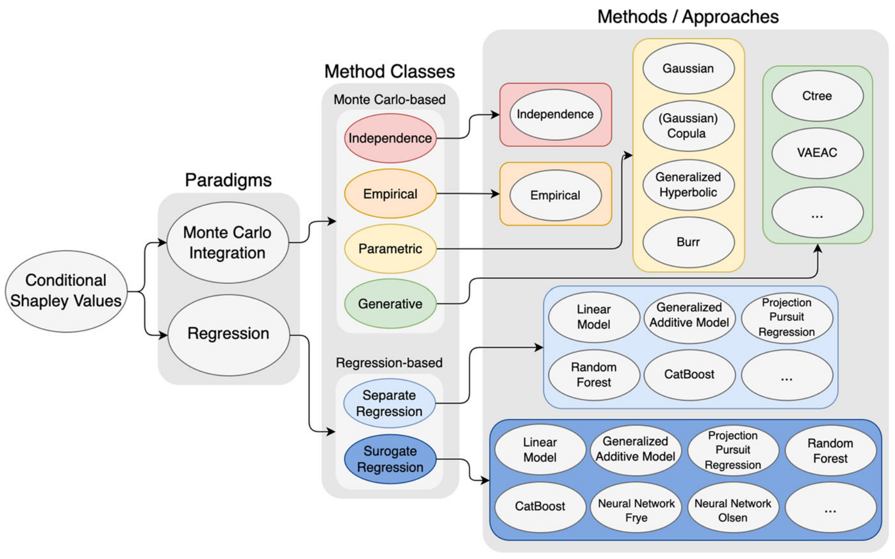

Fig. 1 Schematic overview of the paradigms, method classes, and methods/ approaches used in this article to compute conditional Shapley value explanations

our experiments show that using either a generative method or a regression model with the same form as the predictive model is the best option. In addition to accuracy, we also investigate the computation times of the methods. Based on our findings, we present recommendations for when to use the different method classes and approaches.

In Sect. 2, we give an overview of Shapley values' origin and their use as a model-agnostic explanation framework. The existing and novel methods for estimating the Shapley value explanations are described in Sect. 3. In Sect. 4, we present the simulation studies and discuss the corresponding results. We conduct experiments on real-world data sets in Sect. 5. Recommendations for when to use the different methods and a conclusion are given in Sects. 6 and 7, respectively. In the Appendix, we provide implementation details and more information about some of the methods. We provide additional methods and simulation studies in the Supplement.

2 Shapley values

In this section, we first briefly describe Shapley values in cooperative game theory before we elaborate on their use in model explanation.

2.1 Shapley values in cooperative game theory

Shapley values are a solution concept of how to divide the payout of a cooperative game $v : \mathcal{P}(\mathcal{M}) \mapsto \mathbb{R}$ based on four axioms (Shapley 1953). The game is played by M players where $\mathcal{M} = \{1, 2, \dots, M\}$ denotes the set of all players and $\mathcal{P}(\mathcal{M})$ is the power set, that is, the set of all subsets of $\mathcal{M}$. We call $v(\mathcal{S})$ the *contribution function*[2], and it maps a subset of players $\mathcal{S} \in \mathcal{P}(\mathcal{M})$, also called a coalition, to a real number representing their contribution in the game v. The Shapley values $\phi_j = \phi_j(v)$ assigned to each player j, for $j = 1, \dots, M$, uniquely satisfy the following properties:

Efficiency: They sum to the value of the grand coalition $\mathcal{M}$ over the empty set $\emptyset$, that is, $\sum_{j=1}^{M} \phi_j = v(\mathcal{M}) - v(\emptyset)$.
Symmetry: Two equally contributing players j and k, that is, $v(\mathcal{S} \cup \{j\}) = v(\mathcal{S} \cup \{k\})$ for all $\mathcal{S}$, receive equal payouts $\phi_j = \phi_k$.
Dummy: A non-contributing player j, that is, $v(\mathcal{S}) = v(\mathcal{S} \cup \{j\})$ for all $\mathcal{S}$, receives $\phi_j = 0$.
Linearity: A linear combination of n games $\{v_1, \dots, v_n\}$, that is, $v(\mathcal{S}) = \sum_{k=1}^{n} c_k v_k(\mathcal{S})$, has Shapley values given by $\phi_j(v) = \sum_{k=1}^{n} c_k \phi_j(v_k)$.

Shapley (1953) showed that the values ϕ_j which satisfy these axioms are given by

$$\phi_j = \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{M} \setminus \{j\})} \frac{|\mathcal{S}|!(M - |\mathcal{S}| - 1)!}{M!} \left(v(\mathcal{S} \cup \{j\}) - v(\mathcal{S}) \right), \tag{1}$$

where $|\mathcal{S}|$ is the number of players in coalition $\mathcal{S}$. The number of terms in (1) is 2^M. Hence, the complexity grows exponentially with the number of players M. Each Shapley value is a weighted average of the player's marginal contribution to each coalition $\mathcal{S}$.

2.2 Shapley values in model explanation

We consider the setting of supervised learning where we aim to explain a predictive model $f(\boldsymbol{x})$ trained on $\mathcal{X} = \{\boldsymbol{x}^{[i]}, y^{[i]}\}_{i=1}^{N_{\text{train}}}$, where $\boldsymbol{x}^{[i]}$ is an M-dimensional feature vector, $y^{[i]}$ is a univariate response, and N_{train} is the number of training observations. The prediction $\hat{y} = f(\boldsymbol{x})$, for a specific feature vector $\boldsymbol{x} = \boldsymbol{x}^*$, is explained using Shapley values as a model-agnostic explanation framework (Strumbelj and Kononenko 2010, 2014; Lundberg and Lee 2017). The fairness aspect of Shapley values in the model explanation setting is discussed in, for example, Chen et al. (2020), Fryer et al. (2021), Aas et al. (2021a).

In the Shapley value framework, the predictive model f (indirectly) replaces the cooperative game, and the M-dimensional feature vector replaces the M players. The Shapley value ϕ_j describes the importance of the jth feature in the prediction $f(\boldsymbol{x}^*) = \phi_0 + \sum_{j=1}^{M} \phi_j^*$, where $\phi_0 = \mathbb{E}[f(\boldsymbol{x})]$. That is, the sum of the Shapley values explains the difference between the prediction $f(\boldsymbol{x}^*)$ and the global average prediction.

To calculate (1), we need to define an appropriate contribution function $v(\mathcal{S}) = v(\mathcal{S}, \boldsymbol{x}^*)$ which should resemble the value of $f(\boldsymbol{x}^*)$ when only the features in coalition $\mathcal{S}$ are known. We use the contribution function proposed by Lundberg and Lee (2017), namely the expected response of $f(\boldsymbol{x})$ conditioned on the features in $\mathcal{S}$ taking on the

[2] The $v(\mathcal{S})$ is also called the *reward function* and *characteristic function* in the literature.

values $\boldsymbol{x}_{\mathcal{S}}^*$. That is,

$$\begin{aligned} v(\mathcal{S}) &= \mathbb{E}\left[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \\ &= \mathbb{E}\left[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \\ &= \int f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}}^*) p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)\, \mathrm{d}\boldsymbol{x}_{\bar{\mathcal{S}}}, \end{aligned} \tag{2}$$

where $\boldsymbol{x}_{\mathcal{S}} = \{x_j : j \in \mathcal{S}\}$ denotes the features in subset $\mathcal{S}$, $\boldsymbol{x}_{\bar{\mathcal{S}}} = \{x_j : j \in \bar{\mathcal{S}}\}$ denotes the features outside $\mathcal{S}$, that is, $\bar{\mathcal{S}} = \mathcal{M}\backslash\mathcal{S}$, and $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ is the conditional density of $\boldsymbol{x}_{\bar{\mathcal{S}}}$ given $\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*$. The conditional expectation summarizes the whole probability distribution, it is the most common estimator in prediction applications, and it is also the minimizer of the commonly used squared error loss function (Aas et al. 2021a). Note that the last equality of (2) only holds for continuous features. If there are any discrete or categorical features, the integral should be replaced by sums for these features. Hence, $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ is then no longer continuous.

The contribution function in (2) is also used by, for example, Covert et al. (2020), Aas et al. (2021a), Aas et al. (2021b), Frye et al. (2021), Olsen et al. (2022). Covert et al. (2021) argue that the conditional approach in (2) is the only approach that is consistent with standard probability axioms. Computing (2) is not straightforward for a general data distribution and model. Assuming independent features, or having f be linear, simplifies the computations (Lundberg and Lee 2017; Aas et al. 2021a), but these assumptions do not hold in general.

To compute the Shapley values in (1), we need to compute the contribution function $v(s)$ in (2) for all $\mathcal{S} \in \mathcal{P}(\mathcal{M})$, except for the edge cases $\mathcal{S} \in \{\emptyset, \mathcal{M}\}$. For $\mathcal{S} = \mathcal{M}$, we have that $\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*$ and $v(\mathcal{M}) = f(\boldsymbol{x}^*)$ by definition. For $\mathcal{S} = \emptyset$, we have by definition that $\phi_0 = v(\emptyset) = \mathbb{E}[f(\boldsymbol{x})]$, where the average training response is a commonly used estimate (Aas et al. 2021a). We denote the non-trivial coalitions by $\mathcal{P}^*(\mathcal{M}) = \mathcal{P}(\mathcal{M})\backslash\{\emptyset, \mathcal{M}\}$. The Shapley values $\boldsymbol{\phi}^* = \{\phi_j^*\}_{j=0}^M$ for the prediction $f(\boldsymbol{x}^*)$ are computed as the solution of a weighted least squares problem (Charnes et al. 1988; Lundberg and Lee 2017; Aas et al. 2021a). We refer to Molnar (2023) for an extensive introduction to the Shapley value explanation framework.

In Sects. 2.2.1 and 2.2.2, we describe two prominent paradigms for estimating the contribution function $v(\mathcal{S})$ for all $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, namely, Monte Carlo integration and regression.

2.2.1 Monte Carlo integration

One way to estimate the contribution function $v(\mathcal{S})$ is by using Monte Carlo integration (Lundberg and Lee 2017; Aas et al. 2021a). That is,

$$v(\mathcal{S}) = v(\mathcal{S}, \boldsymbol{x}^*) = \mathbb{E}\left[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \approx \frac{1}{K}\sum_{k=1}^{K} f(\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}, \boldsymbol{x}_{\mathcal{S}}^*) = \hat{v}(\mathcal{S}), \tag{3}$$

where f is the predictive model, $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$, for $k = 1, 2, \ldots, K$, and K is the number of Monte Carlo samples. We insert $\hat{v}(\mathcal{S})$ into (1) to estimate the Shapley values. To obtain accurate conditional Shapley values, we need to generate Monte Carlo samples that follow the true conditional distribution of the data. This distribution is generally not known and needs to be estimated based on the training data. In Sects. 3.1, 3.2, 3.3, and 3.4, we describe different method classes for generating the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$.

2.2.2 Regression

As stated above, the conditional expectation (2) is the minimizer of the mean squared error loss function. That is,

$$\begin{aligned} v(\mathcal{S}) = v(\mathcal{S}, \boldsymbol{x}^*) &= \mathbb{E}\left[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \\ &= \arg\min_c \mathbb{E}\left[(f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}}) - c)^2|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right]. \end{aligned} \tag{4}$$

Thus, any regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$, which is fitted with the mean squared error loss function as the objective function, will approximate (4), obtaining an alternative estimator $\hat{v}(\mathcal{S})$ (Aas et al. 2021a; Frye et al. 2021; Williamson and Feng 2020). The accuracy of the approximation will depend on the form of the predictive model $f(\boldsymbol{x})$, the flexibility of the regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$, and the optimization routine. We can either train a separate regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$ for each $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$ or we can train a single regression model $g(\tilde{\boldsymbol{x}}_{\mathcal{S}})$ which approximates the contribution function $v(\mathcal{S})$ for all $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$ simultaneously. Here $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ is an augmented version of $\boldsymbol{x}_{\mathcal{S}}$ with fixed-length M, where the augmented values are mask values to be explained later. We elaborate on the notation and details of these two regression methodologies in Sects. 3.5 and 3.6, respectively.

2.2.3 Approximation strategies

In general, computing Shapley values is an NP-hard problem (Deng and Papadimitriou 1994; Faigle and Kern 1992), and the complexity of the direct computation of (1) is exponential in the number of features M. In this section, we highlight strategies using approximations or model assumptions to reduce the Shapley value explanation framework's computational complexity and make it tractable in higher dimensions. However, this paper's primary consideration is to accurately compare methods for estimating $v(\mathcal{S})$, but using an approximation strategy will introduce additional uncertainty in the explanations. Thus, we consider relatively low-dimensional settings in the experiments in Sects. 4 and 5, where the exact computation of the Shapley value formula in (1) is feasible. The approximate speed-up strategies can be divided into model-agnostic and model-specific strategies (Chen et al. 2022).

The model-agnostic strategies put no assumptions on the predictive model f and often use stochastic sampling-based estimators (Aas et al. 2021a; Lundberg and Lee 2017; Okhrati and Lipani 2021; Mitchell et al. 2022). That is, to speed up the computations, they approximate the Shapley value explanations by a sampled subset of the coalitions instead of considering the exponential amount of them. Thus, the strategies are stochastic; however, their expectations are unbiased. One of the most common model-agnostic strategies is the `KernelSHAP` strategy introduced in Lundberg and Lee (2017) and improved by Covert and Lee (2021). In the `KernelSHAP` strategy, we sample, e.g., $N_{\mathcal{S}} = 2000 < 2^M$ coalitions and use only these coalitions to approximate the Shapley value explanations. This strategy enables us to approximate the explanations in tractable time even for large values of M; however, a $N_{\mathcal{S}} \ll 2^M$ will (likely) produce poor approximations.

The model-specific strategies put assumptions on the predictive model f to improve the computational cost, but some of the strategies are restricted to marginal Shapley values. For conditional Shapley values, Aas et al. (2021a), Chen et al. (2020) derive explicit expressions for linear models to speed up the computations, and Lundberg et al. (2020) proposes the path-dependent `TreeSHAP` algorithm for tree-based models. There are speed-up strategies for deep neural network models, too, but they are limited to marginal Shapley values (Ancona et al. 2019; Wang et al. 2020).

For more details about the model-agnostic and model-specific strategies, we refer to Chen et al. (2022, Sect. 5.2), which provides an excellent introduction to both strategies. Jethani et al. (2021) proposes another strategy called `FastSHAP`, which sidesteps the Shapley value formula by training a black-box neural network to directly output the Shapley value explanations. Another strategy for reducing the computations is to explain groups of similar/correlated features instead of individual features (Jullum et al. 2021).

3 Conditional expectation estimation

Computing conditional Shapley values is difficult due to the complexity of estimating the conditional distributions, which are not directly available from the training data. In this section, we give a methodological introduction to different methods for estimating the conditional expectation in (2) via either Monte Carlo integration or regression, while we provide implementation details in Appendix A. We organize the methods into six method classes in accordance with those described in Chen et al. (2022, Sect. 5.1.3) and Covert et al. (2021, Sect. 8.2). The method classes we consider are called; `independence`, `empirical`, `parametric`, `generative`, `separate regression`, and `surrogate regression`, and they are described in Sects. 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6, respectively. The first four classes estimate the conditional expectation in (2) using Monte Carlo integration, while the last two classes use regression.

3.1 The independence method

Lundberg and Lee (2017) avoided estimating the complex conditional distributions by implicitly assuming feature independence. In the `independence` approach, the conditional distribution $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$ simplifies to $p(\boldsymbol{x}_{\bar{\mathcal{S}}})$, and the corresponding Shapley values are the marginal Shapley values discussed in Sect. 1. The Monte Carlo samples

$\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}})$ are generated by randomly sampling observations from the training data; thus, no modeling is needed and $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ follows the assumed true data distribution. However, for dependent features, which is common in observational studies, the `independence` approach produces biased estimates of the contribution function (2) and the conditional Shapley values. Thus, the `independence` approach can lead to incorrect conditional Shapley value explanations for real-world data (Aas et al. 2021a; Merrick and Taly 2020; Frye et al. 2021; Olsen et al. 2022).

3.2 The empirical method

Instead of sampling randomly from the training data, the `empirical` method samples only from similar observations in the training data. The optimal procedure is to use only samples that perfectly match the feature values $\boldsymbol{x}_{\mathcal{S}}^*$, as this approach exactly estimates the conditional expectation when the number of matching observations tends to infinity (Chen et al. 2022). However, this is not applicable in practice, as data sets can have few observations, contain a high number of features to match, or have continuous features where an exact match is very unlikely. A natural extension is to relax the perfect match criterion and allow for similar observations (Mase et al. 2019; Sundararajan and Najmi 2020; Aas et al. 2021a). However, this procedure will also be influenced by the curse of dimensionality as conditioning on many features can yield few similar observations and thereby inaccurate estimates of the conditional expectation (2). We can relax the similarity criterion and include less similar observations, but then we break the feature dependencies. The `empirical` approach coincides with the `independence` approach when the similarity measure defines all observations in the training data as similar.

We use the `empirical` approach described in Aas et al. (2021a). The approach uses a scaled version of the Mahalanobis distance to calculate a distance $D_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{[i]})$ between the observation being explained $\boldsymbol{x}^*$ and every training instance $\boldsymbol{x}^{[i]}$. Then they use a Gaussian distribution kernel to convert the distance into a weight $w_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{[i]})$ for a given bandwidth parameter σ. All the weights are sorted in increasing order with $\boldsymbol{x}^{\{k\}}$ having the kth largest value. Finally, they approximate (2) by a weighted version of (3), namely, $\hat{v}(\mathcal{S}) = \sum_{k=1}^{K^*}[w_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{\{k\}}) f(\boldsymbol{x}_{\bar{\mathcal{S}}}^{\{k\}}, \boldsymbol{x}_{\mathcal{S}}^*)] / \sum_{k=1}^{K^*} w_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{\{k\}})$. The number of samples used is $K^* = \min_{L \in \mathbb{N}} \left\{ \sum_{k=1}^{L} w_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{\{k\}}) \big/ \sum_{i=1}^{N_{\text{train}}} w_{\mathcal{S}}(\boldsymbol{x}^*, \boldsymbol{x}^{[i]}) > \eta \right\}$, that is, the ratio between the sum of the K^* largest weights and the sum of all weights must be at least η, for instance, 0.95.

Note that as Aas et al. (2021a) use the Mahalanobis distance, their approach is limited to continuous features. One could potentially extend their method by using a distance measure that supports mixed data, for example, the Gower's distance (Gower 1971; Podani 1999). Another solution is to use, for example, encodings like one-hot-encoding or entity embeddings to represent the categorical variables as numerical (Guo and Berkhahn 2016), although that would increase the computational demand due to increased dimension.

3.3 The parametric method class

In the `parametric` method class, we make a parametric assumption about the distribution of the data. This simplifies the process of generating the conditional Monte Carlo samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$. The idea is to assume a distribution whose conditional distributions have closed-form solutions or are otherwise easily obtainable after estimating the parameters of the full joint distribution. The `parametric` approaches can yield very accurate representations if the data truly follows the assumed distribution, but they may impose a large bias for incorrect parametric assumptions. In this section, we discuss two previously proposed `parametric` approaches and introduce two new methods. The current `parametric` approaches do not support categorical features, which is a major drawback, but one can potentially use the same type of encodings or entity embeddings of the categorical variables as for the `empirical` method.

3.3.1 Gaussian

Both Chen et al. (2020), Aas et al. (2021a) assume that the observations are multivariate Gaussian distributed with mean $\boldsymbol{\mu}$ and covariance matrix Σ. That is, $p(\boldsymbol{x}) = p(\boldsymbol{x}_{\mathcal{S}}, \boldsymbol{x}_{\bar{\mathcal{S}}}) = \mathcal{N}_M(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = [\boldsymbol{\mu}_{\mathcal{S}}, \boldsymbol{\mu}_{\bar{\mathcal{S}}}]^T$ and $\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}} & \boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}} \\ \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\mathcal{S}} & \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\bar{\mathcal{S}}} \end{bmatrix}$. The conditional distribution is also multivariate Gaussian, that is, $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*) = \mathcal{N}_{|\bar{\mathcal{S}}|}(\boldsymbol{\mu}_{\bar{\mathcal{S}}|\mathcal{S}}, \boldsymbol{\Sigma}_{\bar{\mathcal{S}}|\mathcal{S}})$, where $\boldsymbol{\mu}_{\bar{\mathcal{S}}|\mathcal{S}} = \boldsymbol{\mu}_{\bar{\mathcal{S}}} + \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\mathcal{S}}\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}(\boldsymbol{x}_{\mathcal{S}}^* - \boldsymbol{\mu}_{\mathcal{S}})$ and $\boldsymbol{\Sigma}_{\bar{\mathcal{S}}|\mathcal{S}} = \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\bar{\mathcal{S}}} - \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\mathcal{S}}\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}\boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}}$. The parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are easily estimated using the sample mean and covariance matrix of the training data, respectively. In the `Gaussian` approach, we sample the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ from $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$, for $k = 1, 2, \dots, K$ and $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, and use them in (3) to estimate the Shapley values in (1).

3.3.2 Gaussian copula

Aas et al. (2021a) also proposed an alternative approach if the features are far from multivariate Gaussian distributed, namely the (Gaussian) `copula` approach. The idea is to represent the marginals of the features by their empirical distributions and then model the dependence structure by a Gaussian copula. See Appendix B.1 for additional information about copulas.

Assuming a Gaussian copula, Aas et al. (2021a) use the following procedure to generate the K conditional Monte Carlo samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$:

1. Convert each marginal x_j of the feature distribution $\boldsymbol{x}$ to a Gaussian feature v_j by $v_j = \Phi^{-1}(\hat{F}(x_j))$, where $\hat{F}(x_j)$ is the empirical distribution function of marginal j.
2. Assume that $\boldsymbol{v}$ is distributed according to a multivariate Gaussian (the quality of this assumption will depend on how close the Gaussian copula is to the true copula), and sample from the conditional distribution $p(\boldsymbol{v}_{\bar{\mathcal{S}}}|\boldsymbol{v}_{\mathcal{S}} = \boldsymbol{v}_{\mathcal{S}}^*)$ using the method described in Sect. 3.3.1.

3. Convert the margins v_j in the conditional distribution to the original distribution using $\hat{x}_j = \hat{F}_j^{-1}(\Phi(v_j))$.

3.3.3 Burr and generalized hyperbolic

The multivariate Gaussian distribution is probably the most well-known multivariate distribution with closed-form expressions for the conditional distributions. However, any other distribution with easily obtainable conditional distributions is also applicable, for example, the multivariate Burr distribution (Takahasi 1965; Yari and Jafari 2006) and the multivariate generalized hyperbolic (GH) distribution (Barndorff-Nielsen 1977; McNeil et al. 2015; Browne and McNicholas 2015; Wei et al. 2019). We call these two approaches for `Burr` and `GH`, respectively. In contrast to the Gaussian distribution, whose parameters can easily be estimated by the sample means and covariance matrix, the parameters of the Burr and GH distributions are more cumbersome to estimate. We describe the distributions in more detail in Appendix B. The GH distribution is unbounded and can model any continuous data set, while the Burr distribution is strictly positive and is therefore limited to positive data sets. The GH distribution is related to the Gaussian distribution through the t-distribution, where the latter is a special case of the GH distribution and coincides with the Gaussian distribution when the degree of freedom tends to infinity.

3.4 The generative method class

The `generative` and `parametric` methods are similar in that they both generate Monte Carlo samples from the estimated conditional distributions. However, the `generative` methods do not make a parametric assumption about the data. We consider two `generative` approaches: the `ctree` approach of Redelmeier et al. (2020) and the `VAEAC` approach of Olsen et al. (2022). The latter is an extension of the approach suggested by Frye et al. (2021). Both methods support mixed data, i.e., both continuous and categorical data.

3.4.1 Ctree

Redelmeier et al. (2020) compute conditional Shapley values by modeling the dependence structure between the features with conditional inference trees (`ctree`). A `ctree` is a type of recursive partitioning algorithm that builds trees recursively by making binary splits on features until a stopping criterion is satisfied (Hothorn et al. 2006). The process is sequential, where the splitting feature is chosen first using statistical significance tests, and then the splitting point is chosen using any type of splitting criterion. The `ctree` algorithm is independent of the dimension of the response, which in our case is $\boldsymbol{x}_{\bar{\mathcal{S}}}$, while the input features are $\boldsymbol{x}_{\mathcal{S}}$, which varies in dimension based on the coalition $\mathcal{S}$. That is, for each coalition $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, a `ctree` with $\boldsymbol{x}_{\mathcal{S}}$ as the features and $\boldsymbol{x}_{\bar{\mathcal{S}}}$ as the response is fitted to the training data. For a given $\boldsymbol{x}^*_{\mathcal{S}}$, the `ctree` approach finds the corresponding leaf node and samples K observations with replacement from the $\boldsymbol{x}_{\bar{\mathcal{S}}}$ part of the training observations in the same node to generate the conditional Monte Carlo samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$. We get duplicated Monte Carlo samples when K is larger than the number of samples in the leaf node. Thus, the `ctree` method weighs the Monte Carlo samples based on their sampling frequencies to bypass redundant calls to f. Therefore, the contribution function $v(\mathcal{S})$ is not estimated by (3) but rather by the weighted average $\hat{v}(\mathcal{S}) = \sum_{k=1}^{K^*} w_k f(\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}, \boldsymbol{x}_{\mathcal{S}}^*) \big/ \sum_{k=1}^{K^*} w_k$, where K^* is the number of unique Monte Carlo samples. For more details, see Redelmeier et al. (2020, Sect. 3).

3.4.2 VAEAC

Olsen et al. (2022) use a type of variational autoencoder called `VAEAC` (Ivanov et al. 2019) to generate the conditional Monte Carlo samples. Briefly stated, the original variational autoencoder (Kingma and Welling 2014, 2019; Rezende et al. 2014) gives a probabilistic representation of the true unknown distribution $p(\boldsymbol{x})$. The `VAEAC` model extends this methodology to all conditional distributions $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ simultaneously. That is, a single `VAEAC` model can generate Monte Carlo samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ for all coalitions $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$. It is advantageous to only have to fit a single model for all coalitions, as in higher dimensions, the number of coalitions is $2^M - 2$. That is, the number of coalitions increases exponentially with the number of features. In contrast, `ctree` trains $2^M - 2$ different models, which eventually becomes computationally intractable for large M. The `VAEAC` model is trained by maximizing a variational lower bound, which conceptually corresponds to artificially masking features, and then trying to reproduce them using a probabilistic representation. In deployment, the `VAEAC` method considers the unconditional features $\boldsymbol{x}_{\bar{\mathcal{S}}}$ as masked features to be imputed.

3.5 The separate regression method class

The next two method classes use regression instead of Monte Carlo integration to estimate the conditional expectation in (2). In the `separate regression` methods, we train a new regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$ to estimate the conditional expectation for each coalition of features. Related ideas have been explored by Lipovetsky and Conklin (2001), Strumbelj et al. (2009), Williamson and Feng (2020). However, to the best of our knowledge, we are the first to compare different regression models for estimating the conditional expectation as the contribution function $v(\mathcal{S})$ in the local Shapley value explanation framework.

The idea is to estimate $v(\mathcal{S}) = \mathbb{E}\left[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] = \mathbb{E}\left[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right]$ separately for each coalition $\mathcal{S}$ using regression. As in Sect. 2.2, let $\mathcal{X} = \{\boldsymbol{x}^{[i]}, y^{[i]}\}_{i=1}^{N_{\text{train}}}$ denote the training data, where $\boldsymbol{x}^{[i]}$ is the ith M-dimensional input and $y^{[i]}$ is the associated response. For each $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, the corresponding training data set is

$$\mathcal{X}_{\mathcal{S}} = \{\boldsymbol{x}_{\mathcal{S}}^{[i]}, f(\underbrace{\boldsymbol{x}_{\bar{\mathcal{S}}}^{[i]}, \boldsymbol{x}_{\mathcal{S}}^{[i]}}_{\boldsymbol{x}^{[i]}})\}_{i=1}^{N_{\text{train}}} = \{\boldsymbol{x}_{\mathcal{S}}^{[i]}, \underbrace{f(\boldsymbol{x}^{[i]})}_{z^{[i]}}\}_{i=1}^{N_{\text{train}}} = \{\boldsymbol{x}_{\mathcal{S}}^{[i]}, z^{[i]}\}_{i=1}^{N_{\text{train}}}.$$

For each data set $\mathcal{X}_{\mathcal{S}}$, we train a regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$ with respect to the mean squared error loss function. The optimal regression model, with respect to the loss function, is $g^{\dagger}_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \mathbb{E}[z|\boldsymbol{x}_{\mathcal{S}}] = \mathbb{E}[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}}]$, which corresponds to the contribution function $v(\mathcal{S})$. The regression model $g_{\mathcal{S}}$ aims for the optimal $g^{\dagger}_{\mathcal{S}}$. Hence, $g_{\mathcal{S}}$ resembles/estimates the contribution function, that is, $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \hat{v}(\mathcal{S}) \approx v(\mathcal{S}) = \mathbb{E}[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}]$.

A wide variety of regression models minimize the MSE, and we describe a selection of them in Sects. 3.5.1, 3.5.2, 3.5.3, 3.5.4, and 3.5.5. The selection discussed below consists of classical regression models and those that generally perform well for many experiments.

3.5.1 Linear regression model

The simplest regression model we consider is the linear regression model. It takes the form $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \beta_{\mathcal{S},0} + \sum_{j\in\mathcal{S}} \beta_{\mathcal{S},j} x_j = \boldsymbol{x}^T_{\mathcal{S}}\boldsymbol{\beta}_{\mathcal{S}}$, where the coefficients $\boldsymbol{\beta}_{\mathcal{S}}$ are estimated by the least squares solution, that is, $\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \arg\min_{\boldsymbol{\beta}} \|X_{\mathcal{S}}\boldsymbol{\beta} - \boldsymbol{z}\|^2 = (X^T_{\mathcal{S}} X_{\mathcal{S}})^{-1} X^T_{\mathcal{S}}\boldsymbol{z}$, for all $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$. Here $X_{\mathcal{S}}$ is the design matrix with the first column consisting of 1s to also estimate the intercept $\beta_{\mathcal{S},0}$. We call this approach `LM separate`.

3.5.2 Generalized additive model

The generalized additive model (GAM) extends the linear regression model and allows for nonlinear effects between the features and the response (Wood 2006b; Hastie et al. 2009). The fitted GAM takes the form $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \beta_{\mathcal{S},0} + \sum_{j\in\mathcal{S}} g_{\mathcal{S},j}(x_{\mathcal{S},j})$, where the effect functions $g_{\mathcal{S},j}$ are penalized regression splines. We call this approach `GAM separate`.

3.5.3 Projection pursuit regression

The projection pursuit regression (PPR) model extends the GAM model (Friedman and Stuetzle 1981; Hastie et al. 2009). The PPR model takes the form $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}}) = \beta_{\mathcal{S},0} + \sum_{l=1}^{L} g_{\mathcal{S},l}(\boldsymbol{\beta}^T_{\mathcal{S},l}\boldsymbol{x}_{\mathcal{S}})$, where the parameter vector $\boldsymbol{\beta}_{\mathcal{S},l}$ is an $|\mathcal{S}|$-dimensional unit vector. The PPR is an additive model, but in the transformed features $\boldsymbol{\beta}^T_{\mathcal{S},l}\boldsymbol{x}_{\mathcal{S}}$ rather than in the original features $\boldsymbol{x}_{\mathcal{S}}$. The ridge functions $g_{\mathcal{S},l}$ are unspecified and are estimated along with the parameters $\boldsymbol{\beta}_{\mathcal{S},l}$ using some flexible smoothing method. The PPR model combines nonlinear functions of linear combinations, producing a large class of potential models. Moreover, it is a universal approximator for continuous functions for arbitrary large L and appropriate choice of $g_{\mathcal{S},l}$ (Hastie et al. 2009, Sect. 11.2). We call this approach `PPR separate`.

3.5.4 Random forest

A random forest (RF) is an ensemble model consisting of a multitude of decision trees, where the average prediction of the individual trees is returned. The first algorithm was developed by Ho (1995), but Breiman (2001) later extended the algorithm to include bootstrap aggregating to improve the stability and accuracy. We call this approach `RF separate`.

3.5.5 Boosting

A (tree-based) boosted model is an ensemble learner consisting of weighted weak base-learners which has been iteratively fitted to the error of the previous base-learners and together they form a strong learner (Hastie et al. 2009). The seminal boosting algorithm was developed by Freund and Schapire (1997), but multitudes of boosting algorithms has later been developed (Mayr et al. 2014), for example, `CatBoost` (Prokhorenkova et al. 2018). We call this approach `CatBoost separate`.

3.6 The surrogate regression method class

Since the `separate regression` methods train a new regression model $g_{\mathcal{S}}(\boldsymbol{x}_{\mathcal{S}})$ for each coalition $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, a total of $2^M - 2$ models has to be trained, which can be time-consuming for slowly fitted models. The `surrogate regression` method class builds on the ideas from the `separate regression` class, but instead of fitting a new regression model for each coalition, we train a single regression model $g(\tilde{\boldsymbol{x}}_{\mathcal{S}})$ for all coalitions $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$, where $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ is defined in Sect. 3.6.1. The `surrogate regression` idea is used by Frye et al. (2021), Covert et al. (2021), but their setup is limited to neural networks. In Sect. 3.6.1, we propose a general and novel framework that allows us to use any regression model. Then, we relate our framework to the previously proposed neural network setup in Sect. 3.6.2.

3.6.1 General surrogate regression framework

To construct a `surrogate regression` method, we must consider that most regression models g rely on a fixed-length input, while the size of $\boldsymbol{x}_{\mathcal{S}}$ varies with coalition $\mathcal{S}$. Thus, we are either limited to regression models that support variable-length input, or we can create a fixed-length representation $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ of $\boldsymbol{x}_{\mathcal{S}}$ for all coalitions $\mathcal{S}$. The $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ representation must also include fixed-length information about the coalition $\mathcal{S}$ to enable the regression model g to distinguish between coalitions. Finally, we need to augment the training data to reflect that g is to predict the conditional expectation for all coalitions $\mathcal{S}$.

In our framework, we augment the training data by systematically applying all possible coalitions to all training observations. We can then train a single regression model g on the augmented training data set, and the corresponding regression model can then (in theory) estimate the contribution function $v(\mathcal{S})$ for all coalitions $\mathcal{S} \in \mathcal{P}^*(\mathcal{M})$ simultaneously.

To illustrate the augmentation idea, we consider a small example with $M = 3$ features and $N_{\text{train}} = 2$ training observations. Let $\mathcal{X} = \left[\begin{smallmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{smallmatrix}\right]$ and $\boldsymbol{z} = \left[\begin{smallmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \end{smallmatrix}\right]$ denote the training data and responses, respectively. In this setting, $\mathcal{M} = \{1, 2, 3\}$ and $\mathcal{P}^*(\mathcal{M}) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ consists of six different coalitions $\mathcal{S}$, or equivalently masks $\bar{\mathcal{S}} = \mathcal{M}\backslash\mathcal{S}$. Assuming that g relies on fixed-length input, we must represent both the observed values $\boldsymbol{x}_\mathcal{S}$ and coalition $\mathcal{S}$ in a fixed-length notation for the `surrogate regression` methods to work.

To solve this, we first introduce $I(\mathcal{S}) = \{\mathbf{1}(j \in \mathcal{S}) : j = 1, \dots, M\} \in \{0, 1\}^M$, where $\mathbf{1}(j \in \mathcal{S})$ is the indicator function which is one if $j \in \mathcal{S}$ and zero otherwise. Then, $I(\mathcal{S})$ is an M-dimensional binary vector where the jth element $I(\mathcal{S})_j$ is one if the jth feature is in $\mathcal{S}$ (i.e., observed/conditioned on) and zero if it is in $\bar{\mathcal{S}}$ (i.e., unobserved/unconditioned). The I function ensures fixed-length representations of the coalitions/masks, and note that $I(\bar{\mathcal{S}}) = \mathbf{1}^M - I(\mathcal{S})$, where $\mathbf{1}^M$ is the size M vector of 1s. Second, to obtain a fixed-length representation $\hat{\boldsymbol{x}}_\mathcal{S}$ of the observed/conditioned feature vector $\boldsymbol{x}_\mathcal{S}$, we apply the fixed-length mask $I(\mathcal{S})$ to $\boldsymbol{x}$ as an element-wise product, that is, $\hat{\boldsymbol{x}}_\mathcal{S} = \boldsymbol{x} \circ I(\mathcal{S}) = \boldsymbol{x} \circ (\mathbf{1}^M - I(\bar{\mathcal{S}}))$, where $\circ$ is the element-wise product. Finally, we concatenate the fixed-length representations together to form the augmented version of $\boldsymbol{x}_\mathcal{S}$, namely, $\tilde{\boldsymbol{x}}_\mathcal{S} = \{\hat{\boldsymbol{x}}_\mathcal{S}, I(\bar{\mathcal{S}})\}$, which has $2M$ entries. We include $I(\bar{\mathcal{S}})$ in $\tilde{\boldsymbol{x}}_\mathcal{S}$ such that the model g can distinguish between actual zeros in $\boldsymbol{x}_\mathcal{S}$ and those induced by the masking procedure when creating $\hat{\boldsymbol{x}}_\mathcal{S}$. We treat $I(\bar{\mathcal{S}})$ as binary categorical features.

After carrying out this procedure for all coalitions and training observations, we obtain the following augmented training data and responses:

$$\mathcal{X}_{\text{aug}} = \begin{bmatrix} \tilde{\boldsymbol{x}}_{1,\{1\}} \\ \tilde{\boldsymbol{x}}_{1,\{2\}} \\ \tilde{\boldsymbol{x}}_{1,\{3\}} \\ \tilde{\boldsymbol{x}}_{1,\{1,2\}} \\ \tilde{\boldsymbol{x}}_{1,\{1,3\}} \\ \tilde{\boldsymbol{x}}_{1,\{2,3\}} \\ \tilde{\boldsymbol{x}}_{2,\{1\}} \\ \tilde{\boldsymbol{x}}_{2,\{2\}} \\ \tilde{\boldsymbol{x}}_{2,\{3\}} \\ \tilde{\boldsymbol{x}}_{2,\{1,2\}} \\ \tilde{\boldsymbol{x}}_{2,\{1,3\}} \\ \tilde{\boldsymbol{x}}_{2,\{2,3\}} \end{bmatrix} = \left[\begin{matrix} \underbrace{\begin{matrix} x_{11} & 0 & 0 \\ 0 & x_{12} & 0 \\ 0 & 0 & x_{13} \\ x_{11} & x_{12} & 0 \\ x_{11} & 0 & x_{13} \\ 0 & x_{12} & x_{13} \\ x_{21} & 0 & 0 \\ 0 & x_{22} & 0 \\ 0 & 0 & x_{23} \\ x_{21} & x_{22} & 0 \\ x_{21} & 0 & x_{23} \\ 0 & x_{22} & x_{23} \end{matrix}}_{\text{Observed values}} & \underbrace{\begin{matrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}}_{\text{Mask}} \end{matrix}\right] \text{ and } z_{\text{aug}} = \begin{bmatrix} f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_1) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_2) \\ f(\boldsymbol{x}_2) \end{bmatrix}. \qquad (5)$$

The number of rows in $\mathcal{X}_{\text{aug}}$ and z_{aug} is $N_{\text{train}}(2^M - 2)$. For example, with $N_{\text{train}} = 1000$ and $M = 8$ the augmented data $\mathcal{X}_{\text{aug}}$ consists of 254,000 rows, while the number of rows is 65,534,000 when $M = 16$. This exponential growth can make it computationally intractable to fit some types of regression models to the augmented training data $\{\mathcal{X}_{\text{aug}}, z_{\text{aug}}\}$ in high-dimensions. A potential solution is to sample a subset of rows from (5) and only use them to train the regression model. The observation $\boldsymbol{x}^*$, which we want to explain, is augmented by the same procedure, and $g(\tilde{\boldsymbol{x}}^*_\mathcal{S})$ then approximates the corresponding contribution function $v(\mathcal{S}, \boldsymbol{x}^*)$.

Methods

For the `surrogate regression` method class, we consider the same regression models as in Sect. 3.5. We call the methods for `LM surrogate`, `GAM surrogate`, PPR surrogate, RF surrogate, and CatBoost surrogate, and they take the following forms:

LM surrogate: $g(\tilde{\boldsymbol{x}}_{\mathcal{S}}) = \beta_0 + \sum_{j=1}^{2M} \beta_j \tilde{x}_{\mathcal{S},j} = \tilde{\boldsymbol{x}}_{\mathcal{S}}^T \boldsymbol{\beta}$.

GAM surrogate: $g(\tilde{\boldsymbol{x}}_{\mathcal{S}}) = \beta_0 + \sum_{j=1}^{M} g_j(\tilde{x}_{\mathcal{S},j}) + \sum_{j=M+1}^{2M} \beta_j \tilde{x}_{\mathcal{S},j}$. That is, we add nonlinear effect functions to the augmented features $\hat{\boldsymbol{x}}_{\mathcal{S}} = \boldsymbol{x} \circ I(\mathcal{S})$ in $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ while letting the binary mask indicators $I(\bar{\mathcal{S}})$ in $\tilde{\boldsymbol{x}}_{\mathcal{S}}$ be linear.

PPR surrogate: $g(\tilde{\boldsymbol{x}}_{\mathcal{S}}) = \beta_0 + \sum_{l=1}^{L} g_l(\boldsymbol{\beta}_l^T \tilde{\boldsymbol{x}}_{\mathcal{S}})$, where g_l and $\boldsymbol{\beta}_l$ are the lth ridge function and parameter vector, respectively.

RF surrogate: $g(\tilde{\boldsymbol{x}}_{\mathcal{S}})$ is a RF model fitted to the augmented data on the same form as in (5).

CatBoost surrogate: $g(\tilde{\boldsymbol{x}}_{\mathcal{S}})$ is a CatBoost model fitted to the augmented data on the same form as in (5).

3.6.2 Surrogate regression: neural networks

The surrogate regression neural network (NN-Frye surrogate) approach in Frye et al. (2021) differs from our general setup above in that they do not train the model on the complete augmented data. Instead, for each observation in every batch in the training process, they randomly sample a coalition $\mathcal{S}$ with probability $\frac{|\mathcal{S}|!(M-|\mathcal{S}|-1)!}{M!}$. Then they set the masked entries of the observation, i.e., the features not in $\mathcal{S}$, to an off-distribution value not present in the data. Furthermore, they do *not* concatenate the masks to the data, as we do in (5).

We propose an additional neural network (NN-Olsen surrogate) approach to illustrate that one can improve on the NN-Frye surrogate method. The main conceptual differences between the methods are the following. First, for each batch, we generate a missing completely at random (MCAR) mask with paired sampling. MCAR means that the binary entries in the mask $\bar{\mathcal{S}}$ are Bernoulli distributed with probability 0.5, which ensures that all coalitions are equally likely to be considered. Further, paired sampling means that we duplicate the observations in the batch and apply the complement mask, $\mathcal{S}$, on these duplicates. This ensures more stable training as the network can associate both $\boldsymbol{x}_{\mathcal{S}}$ and $\boldsymbol{x}_{\bar{\mathcal{S}}}$ with the response $f(\boldsymbol{x})$. Second, we set the masked entries to zero and include the binary mask entries as additional features, as done in (5) and Olsen et al. (2022). This enables the network to learn to distinguish actual zeros in the data set and zeros induced by the masking, removing the need to set an off-distribution masking value. Additional differences due to implementation, for example, network architecture and optimization routine, are elaborated in Appendix A.

3.7 Time complexity

In this section, we elaborate on the time complexity of computing the Shapley value explanations for the different method classes. We provide a simplified overview of the complexities in Table 1 and the corresponding in-depth explanations here. In general, we can decompose the computation time into three components for each method: *training*, *generating*, and *predicting*. The method-independent computation time of setting up and using the Shapley value explanation framework is described in Appendix A.

Training: The computation time of the training step depends on several method (class) specific attributes. The `independence` method is trivial as no training is needed; hence, its training time is zero. For all other methods, the computation time is affected by the number of features M, training observations N_{train}, and models within the method. The number of models is one for the `VAEAC` and `surrogate regression` methods, while it is $2^M - 2$ for the other methods. In the former case, only a single model is trained, but in return, the `surrogate regression` methods using (5) have doubled M and increased N_{train} by a factor of $2^M - 2$. In the latter case, the number of features in each of the $2^M - 2$ models varies from 1 to M, depending on the coalition $\mathcal{S}$, which also alters the computation time. The computation time is also increased if cross-validation is used to tune some or all of the hyperparameters. Thus, the overall training time complexity is $\mathcal{O}(C_{\text{train}} 2^M)$, where C_{train} is method-specific and depends on the factors discussed above. For example, $C_{\text{train}} = 0$ for the trivial `independence` method. In contrast, estimating a single conditional multivariate normal distribution in the `Gaussian` method yields $C_{\text{train}} = \mathcal{O}(M^2(M + N_{\text{train}}))$, and the same for training a single non-cross-validated linear model in the `LM separate` method. While $C_{\text{train}} = \mathcal{O}(M N_{\text{trees}} N_{\text{train}} \log_2 N_{\text{train}})$ for the `RF separate` method, where N_{trees} is the number of balanced trees in the forest. Thus, the complexities of the four methods are constant, linear, linear, and log-linear in the number of training observations N_{train}, respectively.

Generating: The computation time of the generating step is only applicable to the Monte Carlo-based methods as the regression-based methods do not generate any Monte Carlo samples. The time needed to generate an $|\bar{\mathcal{S}}|$-dimension Monte Carlo sample $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ can vary for different coalitions due to the coalition size. The time complexity of generating the K Monte Carlo samples for the $2^M - 2$ coalitions and N_{test} test observations is $\mathcal{O}(C_{\text{MC}} K N_{\text{test}} 2^M)$, where C_{MC} is method-specific and represents the generation of one Monte Carlo sample. For example, C_{MC} corresponds to sample one observation from the training data in the `independence` method, which is done in constant time, that is, $C_{\text{MC}} = \mathcal{O}(1)$. While in the `Gaussian` method, C_{MC} represents the cost of generating standard Gaussian data and converting it to the associated multivariate conditional Gaussian distribution using the Cholesky decomposition of the conditional covariance matrix $\Sigma_{\bar{\mathcal{S}}|\mathcal{S}}$, which is $\mathcal{O}(M^3)$. Note, however, that the cost of the Cholesky decomposition can be shared among all K Monte Carlos samples and N_{test} test observations. The computation of the Cholesky decomposition could also have been considered part of the training step.

Predicting: The computation time of the predicting step varies between the Monte Carlo and regression paradigms due to their conceptually different techniques for computing $v(\mathcal{S})$. The Monte Carlo paradigm computes the contribution function $v(\mathcal{S})$ based on (3), while the regression paradigm uses (4). The Monte Carlo paradigm relies on averaging K calls to the predictive model f for each of the $(2^M - 2)$ coalition and N_{test} test observation. Thus, the overall predicting time

Table 1 A simplified overview of the time complexities of the paradigm/method classes

Paradigm/Method Class	Training	Generating $\boldsymbol{x}_{\mathcal{S}}^{(k)}$	Predicting $v(\mathcal{S})$
`Monte Carlo`	$\mathcal{O}(C_{\text{train}}2^M)$	$\mathcal{O}(C_{\text{MC}}KN_{\text{test}}2^M)$	$\mathcal{O}(C_fKN_{\text{test}}2^M)$
`Separate regression`	$\mathcal{O}(C_{\text{train}}2^M)$	–	$\mathcal{O}(C_gN_{\text{test}}2^M)$
`Surrogate regression`	$\mathcal{O}(C_{\text{train}})$	–	$\mathcal{O}(C_gN_{\text{test}}2^M)$

See Sect. 3.7 for in-depth explanations. Here, C_{train} is method-specific and depends on, e.g., the number of features M, training observations N_{train}, and if cross-validation is used. Note that 2^M is not applicable for the `VAEAC` method as it is a single model, while the training time of the `independence` method is zero. Furthermore, C_{MC} denotes the computation time of generating one of the K Monte Carlo samples for each of the $2^M - 2$ coalitions and N_{test} test observations. Finally, C_f and C_g represent the computation time of one call to the predictive model f and the regression model g, respectively, and they depend on, e.g., M, N_{train}, and the model complexity

complexity for the Monte Carlo paradigm is $\mathcal{O}(C_fKN_{\text{test}}2^M)$, where C_f represents the computation time of calling f once. In the regression paradigm, the value of the contribution function is directly estimated as the output of a regression model g. Thus, the overall predicting time complexity for the regression paradigm is $\mathcal{O}(C_gN_{\text{test}}2^M)$, where C_g represents the computation time of calling g once. Both C_f and C_g are influenced by, e.g., the number of features M and training observations N_{train}, but also by the intricacy of the predictive and regression model, respectively. For example, the time complexity of calling a linear regression model with M features once is $\mathcal{O}(M)$, while it is $\mathcal{O}(N_{\text{trees}} \log_2 N_{\text{train}})$ for a random forest model with N_{trees} balanced trees.

3.8 Additional methods in the supplement

In addition to the methods described in Sects. 3.1, 3.2, 3.3, 3.4, 3.5, and 3.6, we include dozens more `generative`, `separate regression`, and `surrogate regression` methods in the Supplement. These methods are not included in the main text as they generally perform worse than the introduced methods. For the `generative` method class, we consider three additional `VAEAC` approaches with methodological differences and point to eleven other potential generative methods. For the `separate regression` method class, we consider twenty other regression models, and most of these are also applicable to the `surrogate regression` method class. Among the regression methods are linear regression with interactions, polynomial regression with and without interactions, elastic nets, generalized additive models, principal component regression, partial least squares, K-nearest neighbors, support vector machines, decision trees, boosting, and neural networks. In the Supplement, we apply the additional methods to the numerical simulation studies and real-world data experiments conducted in Sects. 4 and 5, respectively.

4 Numerical simulation studies

A major problem of evaluating explanation frameworks is that there is no ground truth for authentic real-world data. In this section, we simulate data for which we can compute the true Shapley values $\boldsymbol{\phi}_{\texttt{true}}$ and compare how close the estimated Shapley values $\hat{\boldsymbol{\phi}}_{\texttt{q}}$ are when using approach $\texttt{q}$. We gradually increase the complexity of the setups in the simulation studies to uncover in which settings the different methods described in Sect. 3 perform the best and should be used. Additionally, as we focus on conditional Shapley values, we vary the dependencies between the features within each simulation setup to investigate how the methods cope with different dependence levels.

In all experiments, we generate univariate prediction problems with $M = 8$-dimensional features simulated from a multivariate Gaussian distribution $p(\boldsymbol{x}) = \mathcal{N}_8(\mathbf{0}, \Sigma)$, where $\Sigma_{ij} = \rho^{|i-j|}$ for $\rho \in \{0, 0.3, 0.5, 0.9\}$ and 1 on the diagonal. Larger values of ρ correspond to higher dependencies between the features. Higher feature dimensions are possible, but we chose $M = 8$ to keep the computation time of the simulation studies feasible. The real-world data sets in Sect. 5 contain more features. In Sect. 2.2.3, we discuss approximation strategies used in the literature to compute Shapley value explanations in higher dimensions.

We let the number of training observations be $N_{\text{train}} = 1000$, while we explain $N_{\text{test}} = 250$ test observations. Thus, the training data set is $\{\boldsymbol{x}^{[i]}, y^{[i]}\}_{i=1}^{N_{\text{train}}}$, where $\boldsymbol{x}^{[i]} \sim \mathcal{N}_8(\mathbf{0}, \Sigma)$ and the response $y^{[i]} = f_{\texttt{true}}(\boldsymbol{x}^{[i]}) + \varepsilon^{[i]}$. The function $f_{\texttt{true}}$ is different in different experiments and $\varepsilon^{[i]} \sim \mathcal{N}(0, 1)$. The test data sets are created by the same procedure. We provide additional experiments with other settings and some illustrative plots of the data in the Supplement.

We evaluate the performance of the different approaches by computing the mean absolute error (MAE) between the true and estimated Shapley values, averaged over all test observations and features. This criterion has been used in Redelmeier et al. (2020), Aas et al. (2021a), Aas et al. (2021b), Olsen et al. (2022). The MAE is given by

$$\text{MAE} = \text{MAE}_{\phi}(\text{method } \texttt{q}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{1}{M} \sum_{j=1}^{M} |\phi_{j,\texttt{true}}(\boldsymbol{x}^{[i]}) - \hat{\phi}_{j,\texttt{q}}(\boldsymbol{x}^{[i]})|. \quad (6)$$

The true Shapley values are in general unknown, but we can compute them with arbitrary precision in our setup with multivariate Gaussian distributed data, as the conditional distributions $p_{\texttt{true}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$ are analytically known and samplable for all $\mathcal{S}$. Thus, by sampling $\boldsymbol{x}_{\bar{\mathcal{S}},\texttt{true}}^{(k)} \sim p_{\texttt{true}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$, we can compute the true contribution function $v_{\texttt{true}}(\mathcal{S})$ in (2) by using (3). The true Shapley values are then obtained by inserting the $v_{\texttt{true}}(\mathcal{S})$ quantities into the Shapley value formula in (1). The $v_{\texttt{true}}(\mathcal{S})$ quantities can be arbitrarily precise by choosing a sufficiently large number of Monte Carlo samples, e.g., $K = 10{,}000$.

4.1 Linear regression models

The first simulation setup should be considered a sanity check, as we generate the response $y^{[i]}$ according to the following linear regression models:

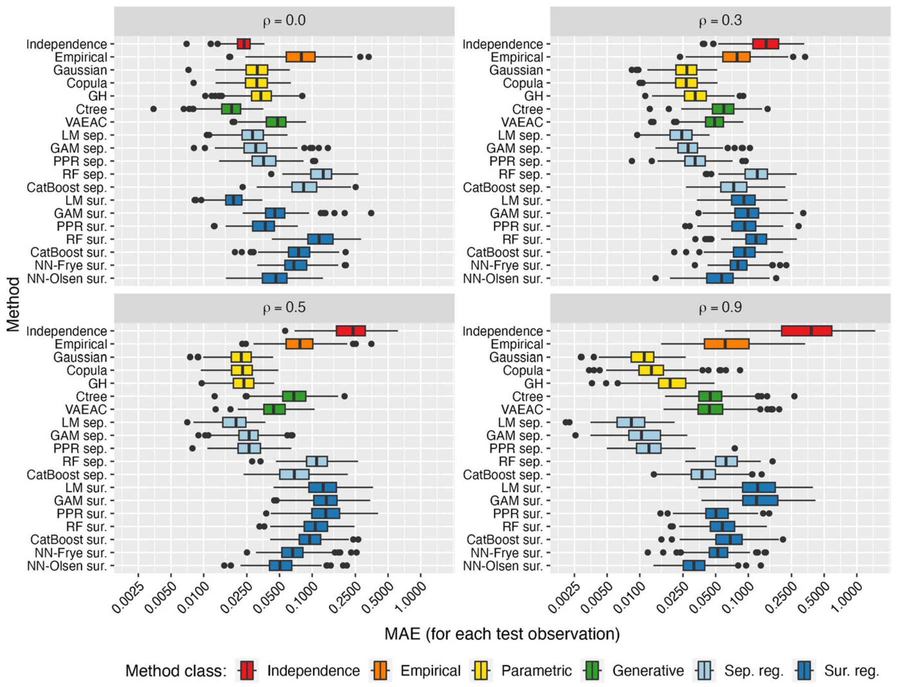

Fig. 2 Results of the `lm_no_interactions` experiment: boxplots of the mean absolute error between the true and estimated Shapley values for the test observations using different methods and for different dependence levels ρ

`lm_no_interactions`: $f_{\text{lm,no}}(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \beta_j x_j$,
`lm_more_interactions`: $f_{\text{lm,more}}(\boldsymbol{x}) = f_{lm,\text{no}}(\boldsymbol{x}) + \gamma_1 x_1 x_2 + \gamma_2 x_3 x_4$,
`lm_numerous_interactions`:
$f_{\text{lm,numerous}}(\boldsymbol{x}) = f_{\text{lm,more}}(\boldsymbol{x}) + \gamma_3 x_5 x_6 + \gamma_4 x_7 x_8$,

where $\boldsymbol{\beta} = \{1.0, 0.2, -0.8, 1.0, 0.5, -0.8, 0.6, -0.7, -0.6\}$ and $\boldsymbol{\gamma} = \{0.8, -1.0, -2.0, 1.5\}$. For each setup, we fit a predictive linear model f with the same form as the true model. For example, in the `lm_more_interactions` setup, the predictive linear model f has eight linear terms and two interaction terms reflecting the form of $f_{\text{lm,more}}$. We fit the predictive models using the `lm` function in base R.

In Figs. 2, 3, and 4, we show the MAE for each test observation (i.e., the absolute error averaged only over the features), and we keep the methods in the same order throughout the figures. In what follows, we briefly summarize the results of the different simulation setups.

`lm_no_interactions` (Fig. 2): For $\rho = 0$, we see that `ctree` and `LM surrogate` perform the best. The `independence` approach, which makes the correct feature independence assumption, is close behind. For $\rho > 0$, the `parametric` and `separate regression` (`LM`, `GAM`, and `PPR`) methods generally perform the best. In particular, the `LM separate` method, which makes the correct model assumption, is the best-performing approach.

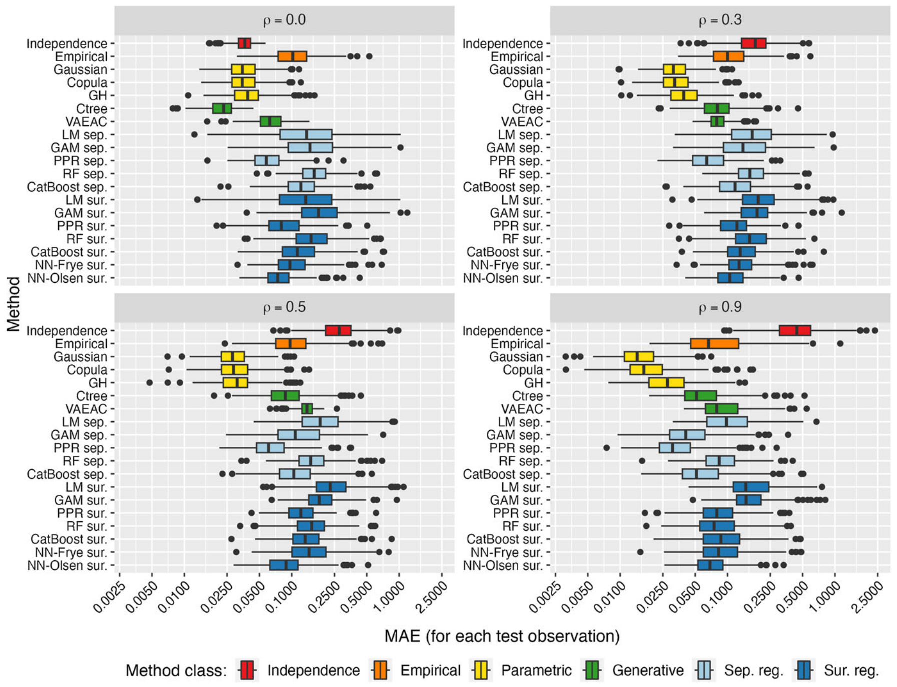

Fig. 3 Results of the `lm_more_interactions` experiment

The `generative` and `empirical` approaches form the mid-field, while the `surrogate regression` and `independence` methods seem to be the least precise.

`lm_more_interactions` (Fig. 3): In this case, the `LM separate` method performs poorly, which is reasonable due to the incorrect model assumption. For $\rho = 0$, the `ctree` approach is the most accurate, but the `independence` and `parametric` methods are close behind. For $\rho > 0$, the `parametric` methods are clearly the best approaches as they make the correct parametric assumption. The `PPR separate` method performs very well, and the `generative` approaches are almost on par for moderate correlation. The `NN-Olsen surrogate` method is the most accurate `surrogate regression` approach. In general, the `separate regression` methods perform better as ρ increases due to the simpler regression problems/prediction tasks. The performance of the `GAM separate` method particularly improves for larger values of ρ.

`lm_numerous_interactions` (Fig. 4): The overall tendencies are very similar to those in the `lm_more_interactions` experiment. The `parametric` methods are by far the most accurate. Further, `ctree` is the best `generative` approach, the `NN-Olsen surrogate` is the best `surrogate regression` method, and the `PPR separate` method is the best `separate regression` approach.

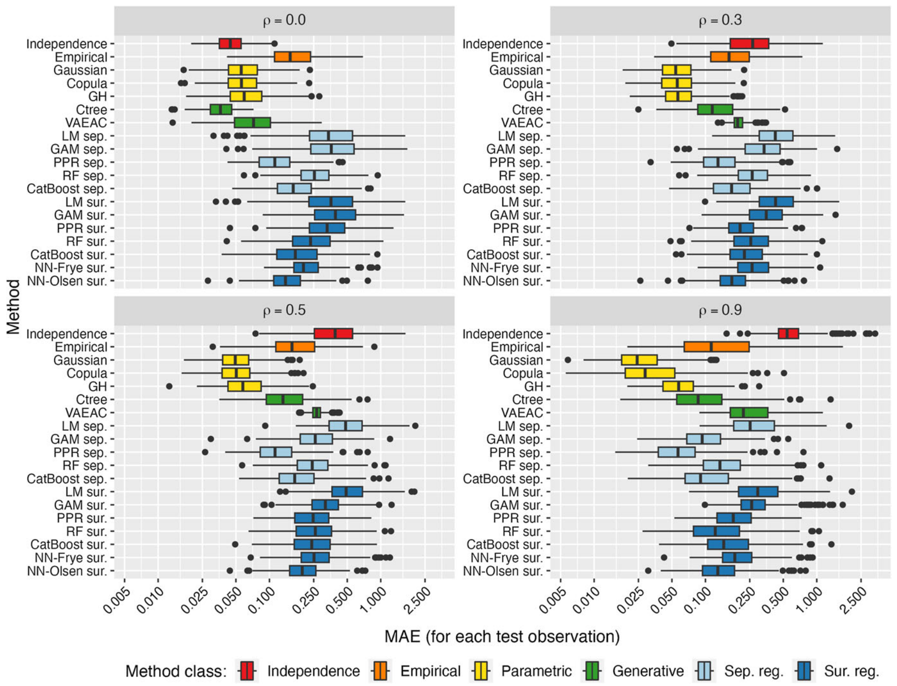

Fig. 4 Results of the `lm_numerous_interactions` experiment

In boxplots, the box and midline represent the interquartile range (IQR) and median, respectively, while the dots denote outliers beyond the whisker boundaries, i.e., observations with MAE scores less/greater than the lower/upper quartile minus/plus 1.5 times the IQR. Olsen (2023) investigates the underlying structure of the outliers and finds that the same test observations often constitute the outliers for the different methods. Crucially, this means that some predictions are intrinsically more challenging to compute accurate Shapley values for. Furthermore, Olsen (2023) illustrates that these observations are often in the outer regions of the training data where less data was available to learn the dependency structures between the features.

4.2 Generalized additive models

In this section, we first investigate situations with a nonlinear relationship between the features and the response, while we later also include pairwise nonlinear interaction terms. More specifically, we first gradually progress from the `lm_no_interactions` model to a full generalized additive model by applying the nonlinear function $\cos(x_j)$ to a subset of the features in $\boldsymbol{x}$. Then, we extend the full generalized additive model by also including pairwise nonlinear interaction terms of the form $g(x_j, x_k) = x_j x_k + x_j x_k^2 + x_k x_j^2$. We generate the features $\boldsymbol{x}^{[i]}$ as before, but the response value $y^{[i]}$ is now generated according to:

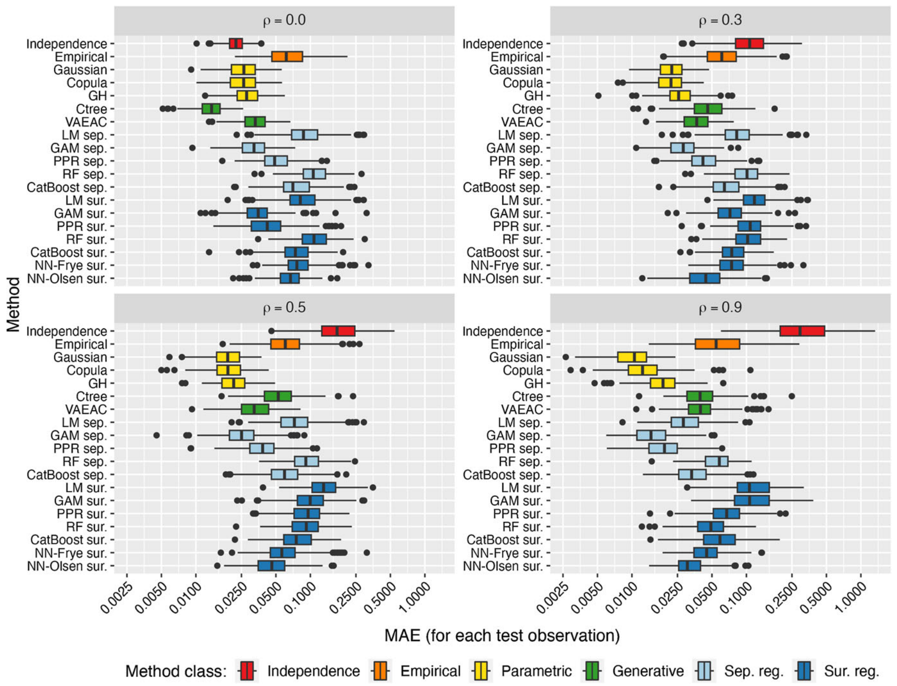

Fig. 5 Results of the `gam_three` experiment

`gam_three`: $f_{\text{gam,three}}(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{3} \beta_j \cos(x_j) + \sum_{j=4}^{M} \beta_j x_j$,
`gam_all`: $f_{\text{gam,all}}(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \beta_j \cos(x_j)$,
`gam_more_interactions`:
$f_{\text{gam,more}}(\boldsymbol{x}) = f_{\text{gam,all}}(\boldsymbol{x}) + \gamma_1 g(x_1, x_2) + \gamma_2 g(x_3, x_4)$,
`gam_numerous_interactions`:
$f_{\text{gam,numerous}}(\boldsymbol{x}) = f_{\text{gam,more}}(\boldsymbol{x}) + \gamma_3 g(x_5, x_6) + \gamma_4 g(x_7, x_8)$,

where we let $\boldsymbol{\beta} = \{1.0, 0.2, -0.8, 1.0, 0.5, -0.8, 0.6, -0.7, -0.6\}$ and $\boldsymbol{\gamma} = \{0.8, -1.0, -2.0, 1.5\}$, i.e., the same coefficients as in Sect. 4.1.

As the true models contain smooth nonlinear effects and smooth pairwise nonlinear interaction terms, we let the corresponding predictive models be GAMs with splines for the nonlinear terms and tensor product smooths for the nonlinear interaction terms. For example, in the `gam_three` experiment, the fitted predictive model f uses splines on the three first features while the others are linear. In the `gam_more_interactions` experiment, f uses splines on all eight features, and tensor product smooths on the two nonlinear interaction terms. We fit the predictive models using the `mgcv` package with default parameters (Wood 2006a, 2022). In what follows, we briefly summarize the results of the different simulation setups.

`gam_three` (Fig. 5): On the contrary to the `lm_no_interactions` experiment, we see that the `LM separate` approach performs much worse than the `GAM separate` approach, which makes sense as we have moved from a linear to a nonlinear setting. For $\rho = 0$, we see that `ctree` and `independence` are the

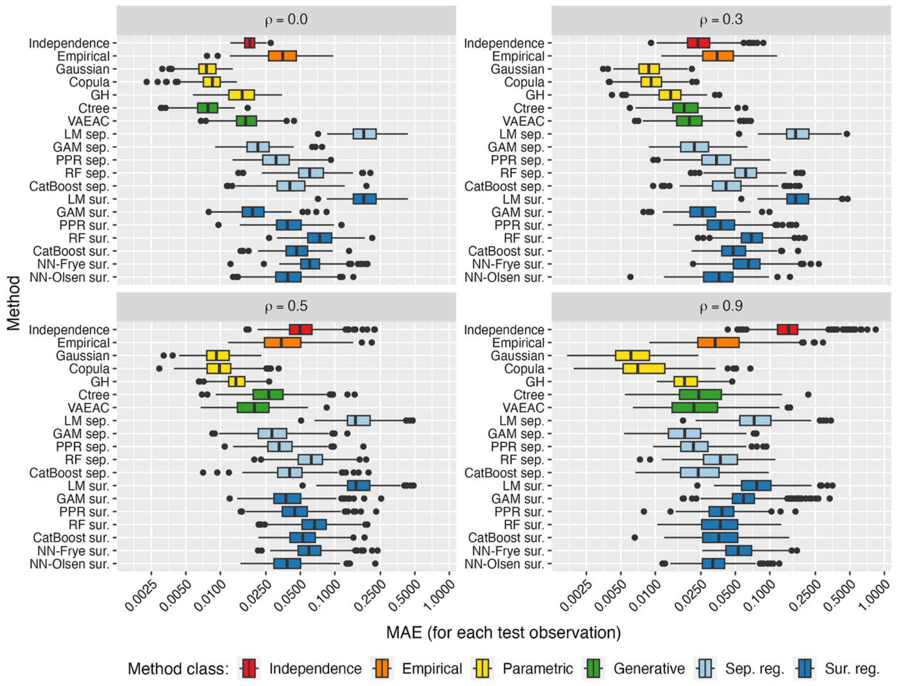

Fig. 6 Results of the `gam_all` experiment

best approaches. For $\rho > 0$, the `parametric` approaches are superior, but the `GAM separate` approach is not far behind, while the `NN-Olsen surrogate` method is the best `surrogate regression` approach.

`gam_all` (Fig. 6): The performance of the `LM` approaches continue to degenerate. The `separate regression` methods get gradually better for higher values of ρ, but the `parametric` methods are still superior. The `generative` methods constitute the second-best class for $\rho \in \{0.3, 0.5\}$, but the `GAM separate` and `PPR separate` approaches are relatively close. The latter approaches outperform the `generative` methods when $\rho = 0.9$.

`gam_more_interactions` (Fig. 7): We see similar results to those in the `gam_all` experiment. The `parametric` approaches are superior in all settings. The `generative` methods perform quite well for $\rho < 0.5$, but they are beaten by the `PPR separate` method for $\rho = 0.9$. Note that the `GAM separate` approach now falls behind the `PPR separate` approach, as it is not complex enough to model the nonlinear interaction terms. This indicates that complex `separate regression` approaches are needed to model complex predictive models. Furthermore, the `RF surrogate` method is on par or outperforms the `NN` based `surrogate regression` approaches.

`gam_numerous_interactions` (Fig. 8): We get nearly identical results as in the previous experiment. Hence, we do not provide further comments on the results.

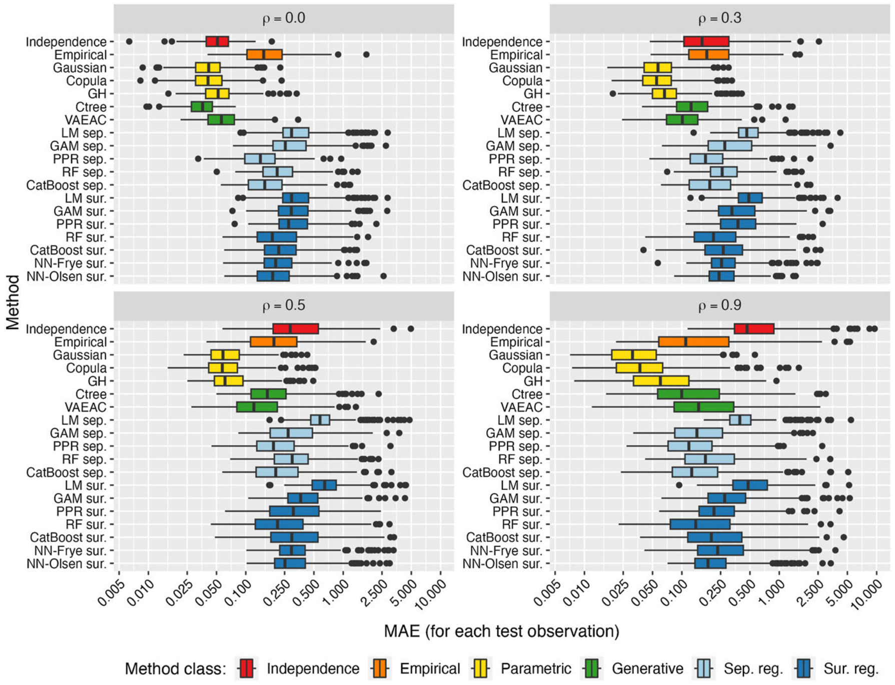

Fig. 7 Results of the `gam_more_interactions` experiment

4.3 Computation time

In this section, we discuss the computation time used by the different methods to estimate the Shapley values, as a proper evaluation of the methods should not only be limited to their accuracy. We report the CPU times to get a fair comparison between the approaches, as some methods are parallelized and would, therefore, benefit from multiple cores when it comes to elapsed time. The CPU times for the different methods will vary significantly depending on the operating system, hardware, and implementation. The times we report here are based on an Intel(R) Core(TM) i5-1038NG7 CPU@2.00GHz with 16GB 3733MHz LPDDR4X RAM running R version 4.2.0 on the macOS Ventura (13.0.1) operating system. Throughout this article, we mean CPU time when we discuss time.

In Table 2, we report the time it took to estimate the Shapley values using the different methods in the `gam_more_interactions` experiment with $\rho = 0.5$, $N_{\text{train}} = 1000$, and $N_{\text{test}} = 250$ in Sect. 4.2. We split the total time into the same three time components as in Sect. 3.7. That is, time used training the approaches, time used generating the Monte Carlo samples, and time used predicting the $v(\mathcal{S})$ using Monte Carlo integration (including the calls to f) or regression. We denote these three components by *training*, *generating*, and *predicting*, respectively. The matrix multiplication needed to estimate the Shapley values from the estimated contribution functions is almost instantaneous and is part of the predicting time. Furthermore,

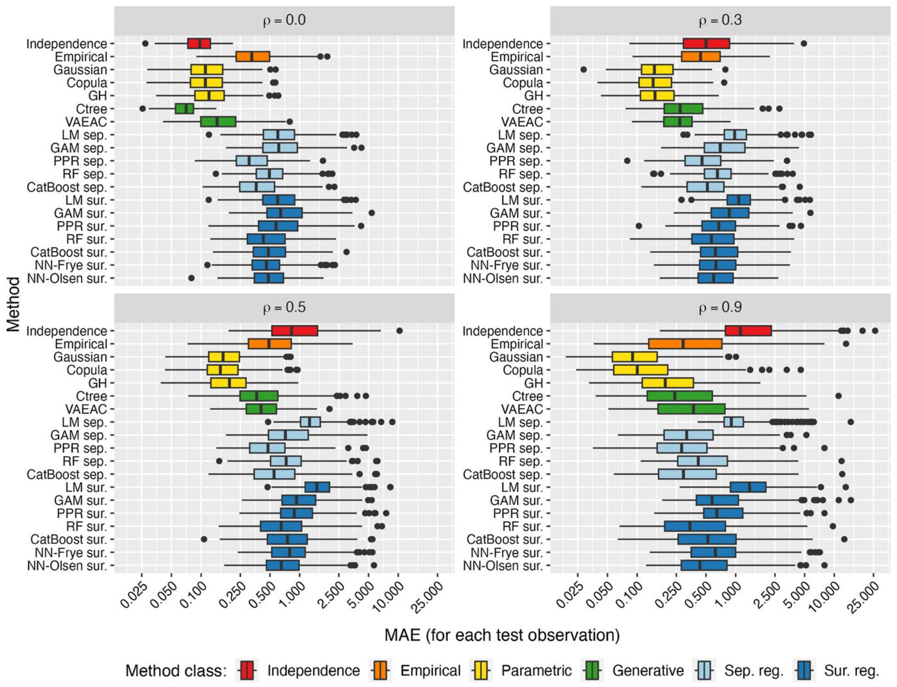

Fig. 8 Results of the `gam_numerous_interactions` experiment

Table 2 The CPU times used by the methods to compute Shapley values for the $N_{\text{test}} = 250$ test observations in the `gam_more_interactions` experiment with $\rho = 0.5$ and $N_{\text{train}} = 1000$

Method	Training	Generating $\boldsymbol{x}_{\mathcal{S}}^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time
`Independence`	0.0	4.6	35:23.0	35:27.6
`Empirical`	4.4	19.0	8:47.6	9:11.0
`Gaussian`	0.0	1:22.7	35:36.3	36:59.0
`Copula`	0.0	5:53.5	35:18.0	41:11.5
`GH`	2:05.3	1:34.7	36:15.8	39:55.8
`Ctree`	3.2	3:47.1	10:31.1	14:21.4
`VAEAC`	56.3	3:07.7	34:41.1	38:45.1
`LM sep.`	0.3	—	0.2	0.5
`GAM sep.`	34.6	—	4.8	39.4
`PPR sep.`	1:39.2	—	0.3	1:39.5
`RF sep.`	58:46.0	—	5.3	58:51.3
`CatBoost sep.`	5:44.8	—	0.1	5:44.9
`LM sur.`	2.3	—	0.4	2.7
`GAM sur.`	12.5	—	8.6	21.1
`PPR sur.`	3:49.5	—	0.5	3:50.0
`RF sur.`	1:05:55.5	—	7.9	1:06:03.4
`CatBoost sur.`	38.8	—	0.4	39.2
`NN-Frye sur.`	13:56:43.9	—	1.8	13:56:45.7
`NN-Olsen sur.`	7:31:43.8	—	1.9	7:31:45.7

The format of the CPU times is hours:minutes:seconds, where we omit the larger units of time if they are zero, and the colors indicate the different method classes creating the augmented training data for the `surrogate regression` methods in Sect. 3.6.1 takes around one second and is part of the training time. We see a wide spread in the times, but, in general, the Monte Carlo approaches take, on average, around half an hour, while the regression methods are either much faster or slower, depending on the approach.

The Monte Carlo methods make a total of $N_{\text{test}}K(2^M - 2)$ calls to the predictive model f to explain the N_{test} test observations with M features and K Monte Carlo samples. In our setting with $N_{\text{test}} = 250$, $M = 8$, and $K = 250$, the predictive model is called N_f =15,875,000 times, thus, the speed of calculating f greatly effects the explanation time. For example, the GAM model in the `gam_more_interactions` experiment is slow, as we can see in Table 2, since the predicting time constitutes the majority of the total time. To compare, N_f calls to the linear model in the `lm_more_interactions` experiment takes approximately 3 CPU seconds, while the GAMs in the `gam_three` and `gam_more_interactions` experiments take roughly 13 and 35 CPU minutes, respectively. In the latter experiment, the PPR and RF models in Sect. 4.5 take around 0.5 and 40 CPU minutes, respectively.

The `empirical` and `ctree` approaches have lower predicting time than the other Monte Carlo-based methods due to fewer calls to f since they use weighted Monte Carlo samples; see Sects. 3.2 and 3.4.1. The three influential time factors for the Monte Carlo methods are: the training time of the approach (estimating the parameters), the sampling time of the Monte Carlo samples, and the computational cost of calling f; see Sect. 3.7.

In contrast, both the `separate regression` and `surrogate regression` methods use roughly the same time to estimate the Shapley values for different predictive models f, as f is only called N_{train} times when creating the training data sets. After that, we train the `separate regression` and `surrogate regression` approaches and use them to directly estimate the contribution functions. The influential factors for the regression methods are the training time of the $2^M - 2$ separate models (or the one surrogate model) and the prediction time of calling them a total of $N_{\text{test}}(2^M - 2)$ times. The former is the primary factor, and it is influenced by, e.g., hyperparameter tuning and the training data size. The latter can be a problem for the augmented training data for the `surrogate regression` methods, as we will see in Sect. 5.4.

When excluding the time of the training step, which is only done once and can be considered as an upfront time cost, it is evident that the regression-based methods produce the Shapley value explanations considerably faster than the Monte Carlo-based methods. For example, consider the most accurate Monte Carlo and regression-based methods in the `gam_more_interactions` experiment with $\rho = 0.5$, i.e., the `Gaussian` and `PPR separate` methods, respectively. The `Gaussian` approach uses approximately 37 CPU minutes to explain 250 predictions, an average of 8.88 seconds per explanation. In contrast, the `PPR separate` method explains all the $N_{\text{test}} = 250$ predictions in half a second. Thus, the `PPR separate` method is approximately 4440 times faster than the `Gaussian` approach per explanation, which is essential for large values of N_{test}. However, note that this factor is substantially lower for predictive models that are less computationally expensive to call.

The computation times reported in Table 2 align with the time complexities discussed in Sect. 3.7. However, it is crucial to consider the cost of the cross-validation procedure when evaluating the computation times. For example, the `RF separate` method is by far the slowest `separate regression` method, but the large training time is (mainly) due to the method's extensive hyperparameter tuning described in Appendix A. As the number of folds and hyperparameter combinations in the cross-validation procedure is $N_{\text{folds}} = 4$ and $N_{\text{hyper}} = 12$, respectively, we fit a total of $N_{\text{folds}} N_{\text{hyper}}(2^M - 2) = 12\,192$ random forest models. In contrast, the `LM separate` method directly fits the $2^M - 2$ separate models without any tuning. In the Supplement, we omit the cross-validation procedure and use default hyperparameter values, which significantly reduces the computation time but also the performance. The two slowest methods are the `NN-Frye` and `NN-Olsen surrogate` methods, which consider six and nine hyperparameter combinations each. Thus, using default values would reduce the training time by a factor of 6 and 9, respectively, but at the cost of precision.

4.4 Number of training observations

To investigate if the training data size has an effect on the ordering of the methods according to the MAE criterion, we repeat the experiments in Sects. 4.1 and 4.2 with $N_{\text{train}} \in \{100, 5000\}$, and some of them with $N_{\text{train}} = 20{,}000$. We obtain nearly identical results, except for three distinctions. First, the `independence` approach becomes relatively more accurate compared to the other methods when $N_{\text{train}} = 100$, and worse when $N_{\text{train}} \in \{5000, 20, 000\}$. This is intuitive, as modeling the data distribution/response is easier when the methods have access to more data. Second, in the simple experiments in Sects. 4.1 and 4.2 and $N_{\text{train}} \in \{5000, 20, 000\}$, the `GAM separate` and `PPR separate` approaches become even better, but are still beaten by the `Gaussian` and `copula` approaches in most experiments. Third, we observe that the MAE has a tendency to decrease when N_{train} increases. However, we cannot directly compare the MAE scores as they depend on the fitted predictive model f, which changes when N_{train} is adjusted.

4.5 Other choices for the predictive model

In practice, it might be difficult to identify the pairwise interactions in Sect. 4.2. Hence, one would potentially fit a model without them. We included them above as we knew the data-generating processes and wanted a precise model, but we now pretend otherwise and fit other predictive models. We consider two different types of complex black-box predictive models: projection pursuit regression (PPR) and random forest (RF), and we conduct the same type of hyperparameter tuning as in the other experiments. However, we conduct no feature transformations and directly use the original features when fitting the models. These models are less precise than the GAMs in Sect. 4.2, which have an unfair advantage as they use the true formulas. For example, in the `gam_more_interactions` experiment, the MSE test prediction values were

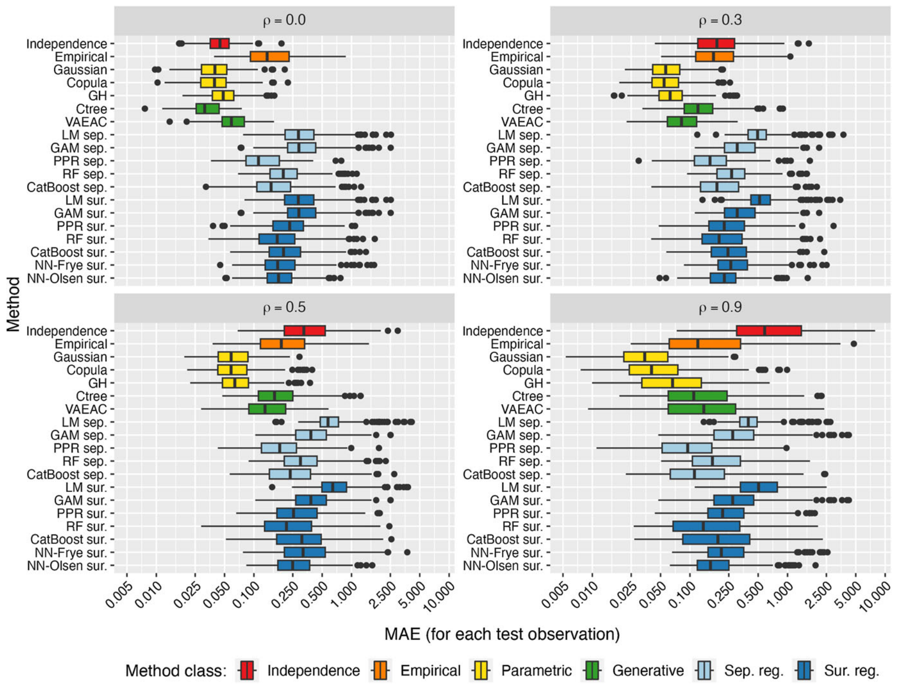

Fig. 9 Experiment `gam_more_interactions` with a PPR as the predictive model

1.32, 3.67, 7.36 for GAM, PPR, RF, respectively, where 1 is the theoretical optimum as $\mathrm{Var}(\varepsilon) = 1$.

We only include the figures for the `gam_more_interactions` experiment, as the corresponding figures for the other experiments are almost identical. The results are displayed in Figs. 9 and 10 for the PPR and RF models, respectively, and the results are quite similar to those obtained for the GAM model in Fig. 7. In general, the `parametric` methods are superior, followed by the `generative` methods, while the `empirical`, `separate regression`, and `surrogate regression` approaches are worse. Some `separate regression` approaches perform, however, much better for high dependence. The `independence` method performs well when $\rho = 0$, but it gradually degenerates as the dependence level increases, as expected. We see that the `PPR separate` approach performs well for the PPR predictive model, but it is outperformed by the `CatBoost separate` method for the RF models. These results indicate that for our experiments, it is beneficial to choose a regression method similar to the predictive model; that is, for a non-smooth model, one should consider using a non-smooth regression method. However, note that the difference in the MAE is minuscule.

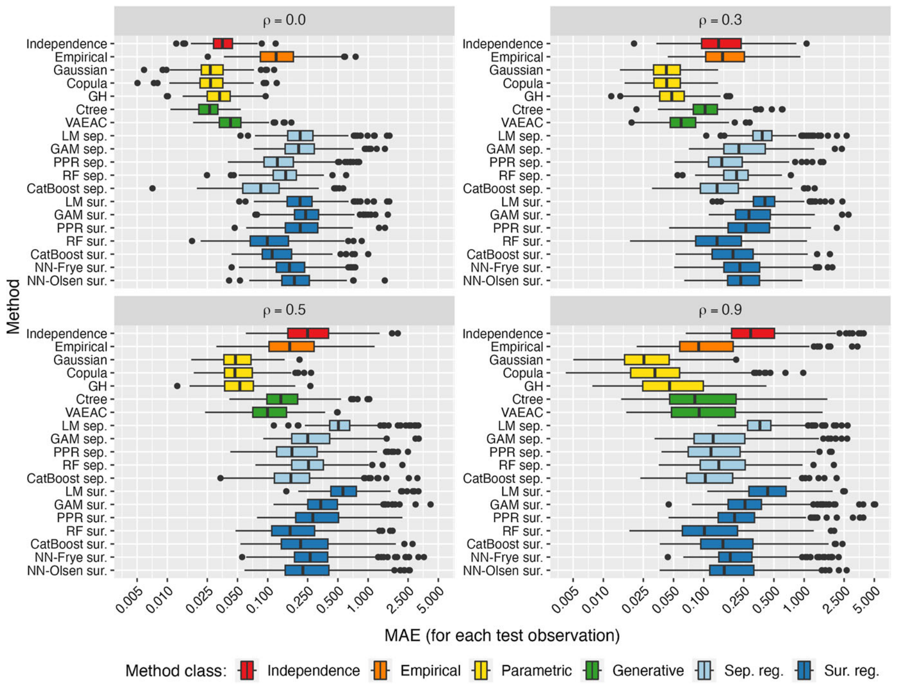

Fig. 10 Experiment `gam_more_interactions` with a RF as the predictive model

4.6 Different data distribution

In the Supplement, we repeat all the experiments described in Sects. 4.1 and 4.2 but with multivariate Burr distributed features instead of Gaussian ones. The Burr distribution allows for heavy-tailed, skewed marginals, and nonlinear dependence. In this case, the `parametric Burr` approach, which assumes Burr distributed data, not surprisingly, is the most accurate. The `Gaussian` method, which now incorrectly assumes Gaussian distributed data, performs worse. The `VAEAC` approach performs very well on the Burr distributed data, which was also observed by Olsen et al. (2022). In general, `VAEAC` is the second-best approach after `Burr`. The `PPR separate` method also performs well, but compared to the `Burr` and `VAEAC` approaches, it is less precise in the experiments with nonlinear interaction terms.

4.7 Summary of the experiments

Making the correct (or nearly correct) parametric assumption about the data is advantageous, as the corresponding `parametric` methods significantly outperform the other approaches in most settings. In general, if the distribution is unknown, the second-best option for low to moderate levels of dependence is the `generative` method class. The `separate regression` approaches improve relative to the other methods when the feature dependence increases, and for highly dependent features, the `PPR separate` approach is a prime choice. Furthermore, the `separate regression` methods that match the form of f often give more accurate Shapley value estimates. The PPR model in the `PPR separate` approach is simple to fit but is still very flexible and can, therefore, accurately model complex predictive models. The `independence` approach is accurate for no (or very low) feature dependence, but it is often the worst approach for high feature dependence. The `NN-Olsen surrogate` method outperforms the `NN-Frye surrogate` approach in most settings and is generally the best `surrogate regression` approach.

We found it (often) necessary to conduct some form of cross-validation to tune (most of) the `separate regression` and `surrogate regression` methods to make them more competitive. Using default hyperparameter values usually resulted in less accurate Shapley value explanations; see additional experiments in the Supplement. The hyperparameter tuning can be time-consuming, but it was feasible in our setting with $M = 8$ features and $N_{\text{train}} = 1000$ training observations. The regression-based methods use most of their computation time on training, while the predicting step is almost instantaneous for several methods. The opposite holds for the Monte Carlo-based approaches, which are overall slower than most regression-based methods. Hence, we have a trade-off between computation time and Shapley value accuracy in the numerical simulation studies. We did not conduct hyperparameter tuning for the `empirical`, `parametric`, and `generative` methods. Thus, the methods where we conduct hyperparameter tuning have an unfair advantage regarding the precision of the estimated Shapley values.

5 Real-world data experiments

In this section, we fit several predictive models to different real-world data sets from the UCI Machine Learning Repository and then use the Shapley value explanation framework to explain the models' predictions. The models range from complex statistical models to black-box machine learning methods. We consider four data sets: Abalone, Diabetes, Wine, and Adult. Some illustrative data plots are provided in the Supplement.

For real-world data sets, the true Shapley values are unknown. Hence, we cannot use the MAE evaluation criterion from Sect. 4 to evaluate and rank the approaches. Instead, we use the MSE_v criterion proposed by Frye et al. (2021) and later used by Olsen et al. (2022). The MSE_v is given by

$$\text{MSE}_v = \text{MSE}_v(\text{method } \texttt{q}) = \frac{1}{N_{\mathcal{S}}} \sum_{\mathcal{S} \in \mathcal{P}^*(\mathcal{M})} \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left(f(\boldsymbol{x}^{[i]}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}^{[i]}) \right)^2, \tag{7}$$

where $N_{\mathcal{S}} = |\mathcal{P}^*(\mathcal{M})| = 2^M - 2$ and $\hat{v}_{\texttt{q}}$ is the estimated contribution function using method $\texttt{q}$. The motivation behind the MSE_v criterion is that $\mathbb{E}_{\mathcal{S}}\mathbb{E}_{\boldsymbol{x}}(v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}))^2$ can be decomposed as

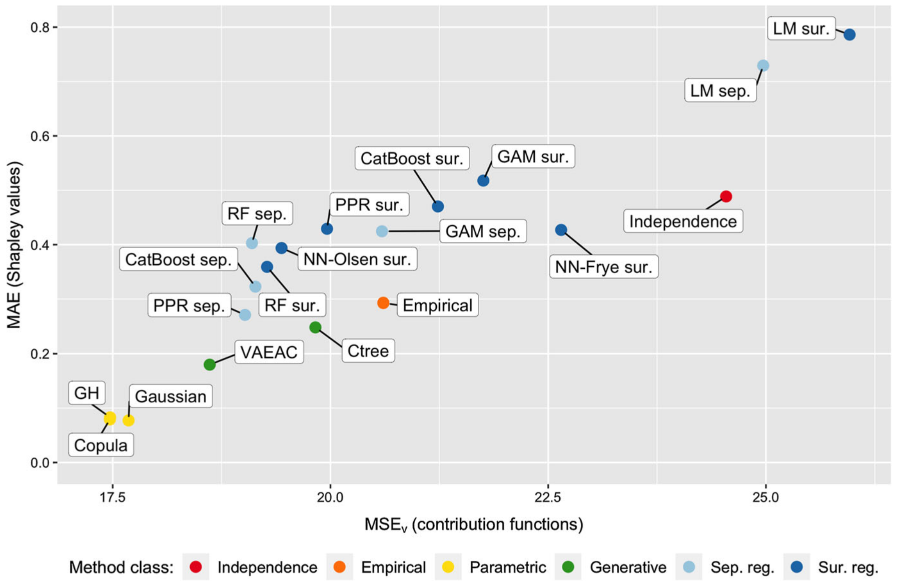

Fig. 11 Illustration of a fairly strong linear relationship between the MSE$_v$ and MAE criteria for the `gam_more_interactions` experiment with Gaussian data and $\rho = 0.5$

$$\mathbb{E}_\mathcal{S}\mathbb{E}_{\boldsymbol{x}}(v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}))^2 = \mathbb{E}_\mathcal{S}\mathbb{E}_{\boldsymbol{x}}(f(\boldsymbol{x}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}))^2 - \mathbb{E}_\mathcal{S}\mathbb{E}_{\boldsymbol{x}}(f(\boldsymbol{x}) - v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}))^2, \tag{8}$$

see Covert et al. (2020, Appendix A). The first term on the right-hand side of (8) can be estimated by (7), while the second term is a fixed (unknown) constant not influenced by the approach $\texttt{q}$. Thus, a low value of (7) indicates that the estimated contribution function $\hat{v}_{\texttt{q}}$ is closer to the true counterpart $v_{\texttt{true}}$ than a high value.

An advantage of the MSE$_v$ criterion is that $v_{\texttt{true}}$ is not involved. Thus, we can apply it to real-world data sets. However, the criterion has two drawbacks. First, we can only use it to rank the methods and not assess their closeness to the optimum since the minimum value of the MSE$_v$ criterion is unknown. Second, the criterion evaluates the contribution functions and not the Shapley values. It might be the case that the estimates for $v(\mathcal{S})$ overshoot for some coalitions and undershoot for others, and such errors may cancel each other out in the Shapley value formula in (1). Nevertheless, in the numerical simulation studies in Sect. 4, we computed both criteria to compare the ordering of two criteria empirically. We generally observe a relatively linear relationship between the MAE and MSE$_v$ criteria. That is, a method that achieves a low MSE$_v$ score also tends to obtain a low MAE score, and vice versa. To illustrate this tendency, we include Fig. 11, where we plot the MSE$_v$ criterion against the MAE criterion for the `gam_more_interactions` experiment with Gaussian distributed data with $\rho = 0.5$. Note that the orderings of the two criteria are not one-to-one, but they give fairly similar rankings of the methods.

In Fig. 3, we report the MSE_v scores and CPU times of the different methods for the four data sets. The Abalone, Diabetes, and Wine data sets were run on the same system as specified in Sect. 4.3, while the Adult data set was run on a shared computer server running Red Hat Enterprise Linux 8.5 with two Intel(R) Xeon(R) Gold 6226R CPU@2.90GHz (16 cores, 32 threads each) and 768GB DDR4 RAM, due to memory constraints on the former system. Thus, one should not compare the CPU times across these systems but only the CPU times of the different methods within the same experiment. More detailed decomposition of the CPU times and additional methods are provided in the Supplement.

5.1 Abalone

We first consider the classical Abalone data set with mixed features. The data set originates from a study by the Tasmanian Aquaculture and Fisheries Institute (Nash et al. 1994) and has been used in several XAI papers (Vilone et al. 2020; Aas et al. 2021b; Frye et al. 2021; Olsen et al. 2022). The data set contains clear nonlinearity and heteroscedasticity among the pairs of features, and there is a significant pairwise correlation between the features, as all continuous features have a pairwise correlation above 0.775. The mean correlation is 0.89, and the maximum is 0.99. Furthermore, all marginals are skewed.

We split the 4177 observations into training (75%) and testing (25%) data sets. The goal is to predict the age of the abalone based on $M = 8$ easily obtainable features: `Length`, `Diameter`, `Height`, `WholeWeight`, `ShuckedWeight`, `VisceraWeight`, `ShellWeight`, and `Sex`. All features are continuous except for `Sex` which is a three-level categorical feature (infant, female, male). Thus, the `empirical` and `parametric` methods are not applicable. However, to remedy this, we train two PPR models to act as our predictive models; one based on all features (PPR_{all}) and another based solely on the continuous features (PPR_{cont}). We chose the PPR model as it outperformed the other prediction models we fitted (GAM, RF, CatBoost). The test MSE increases from 2.04 to 2.07 when excluding `Sex`. Cross-validation determined that number of terms in PPR_{all} and PPR_{cont} should be 4 and 7, respectively.

Table 3 shows that the best approaches for explaining the PPR predictive models are the `PPR separate`, `NN-Olsen surrogate`, and `VAEAC` methods. For the $\text{Abalone}_{\text{cont}}$ data set, the `PPR separate` and `NN-Olsen surrogate` methods perform equally well and share first place, but both methods are marginally outperformed by the `VAEAC` approach for the $\text{Abalone}_{\text{all}}$ data set. However, both the `VAEAC` and `NN-Olsen surrogate` methods are very slow compared to the `PPR separate` approach. The second-best Monte Carlo-based method for the $\text{Abalone}_{\text{cont}}$ data set is the Gaussian `copula` approach, even though the Abalone data set is far from Gaussian distributed. This is probably because the `copula` method does not make a parametric assumption about the marginal distributions of the data, but rather the copula/dependence structure, which makes it a more robust method than the `Gaussian` approach.

Table 3 The MSE_v scores and CPU times for the methods applied to the real-world data sets in Sect. 5

	Abalone$_{\text{cont}}$ (M=7)		Abalone$_{\text{all}}$ (M=8)		Diabetes (M=10)		Wine (M=11)		Adult (M=14)	
Method	MSE_v	Time	MSE_v	Time	MSE_v	Time	MSE_v	Time	MSE_v	Time
Independence	8.679	1:24.2	9.144	3:51.5	0.196	38.4	0.145	4:00:53.5	0.041	1:10:36.3
Empirical	1.540	3:43.2	—	—	0.143	15.1	0.088	2:33:18.5	—	—
Gaussian	1.349	3:44.0	—	—	0.127	2:35.1	0.118	4:08:28.6	—	—
Copula	1.223	15:05.5	—	—	0.127	10:54.5	0.107	4:53:36.6	—	—
GH	1.292	8:39.7	—	—	0.133	7:31.3	0.109	4:23:39.9	—	—
Burr	5.640	5:22.3	—	—	—	—	0.202	3:52:32.7	—	—
Ctree	1.393	7:40.8	1.424	19:14.1	0.158	6:46.9	0.102	1:02:41.5	—	—
VAEAC	1.182	2:34:03.6	**1.180**	11:48:22.1	0.128	21:57.8	0.093	6:09:37.9	0.027	5:12:31:03.7
LM sep.	1.684	0.3	1.581	0.9	**0.126**	1.9	0.146	4.6	0.043	9:00:13.5
GAM sep.	1.298	33.7	1.299	1:09.7	**0.126**	1:03.6	0.124	3:24.6	0.033	2:38:52.6
PPR sep.	**1.169**	2:15.1	1.185	3:34.6	**0.126**	5:22.4	0.129	25:19.4	0.032	14:12:19:47.6
RF sep.	1.239	1:09:15.8	1.259	2:31:09.7	0.143	1:00:23.6	**0.071**	9:21:12.9	0.027	98:13:33:27.4
CatBoost sep.	1.190	6:17.1	1.213	18:25.0	0.135	18:40.6	0.082	1:41:32.7	**0.026**	35:09:59:59.1
LM sur.	2.912	3.7	2.770	7.2	0.165	4.4	0.162	25.9	—	—
GAM sur.	2.611	41.1	2.557	1:24.6	0.168	21.3	0.145	4:01.4	—	—
PPR sur.	1.548	14:57.5	1.538	55:31.6	0.136	4:13.0	0.149	1:20:15.5	—	—
RF sur.	1.281	1:14:30.8	1.311	3:46:34.1	0.143	1:33:51.3	0.085	1:15:42:46.3	—	—
CatBoost sur.	1.298	9:10.8	1.348	29:28.6	0.140	53.4	0.108	29:06.4	—	—
NN-Frye sur.	1.244	3:01:48:38.7	1.320	3:16:01:07.5	0.154	3:11:39.4	0.170	1:10:53:27.4	0.085	3:16:13:50.3
NN-Olsen sur.	**1.169**	3:22:23:46.0	1.192	2:01:07:30.3	0.135	1:28:54.8	0.130	23:56:47.0	0.045	3:11:33:57.9

The format of the CPU times is days:hours:minutes:seconds, where we omit the larger units of time if they are zero, and the colors indicate the different method classes

5.2 Diabetes

The diabetes data set stems from Efron et al. (2004) and contains $M = 10$ baseline features; Age, Sex, BMI, BP (blood pressure), and six blood serum measurements (S1, S2, S3, S4, S5, S6) obtained from 442 diabetes patients. The response of interest is a quantitative measure of disease progression one year after the baseline. Like Efron et al. (2004), we treat Sex as numerical and standardize all features; hence, we can apply all methods. Many features are strongly correlated, with a mean absolute correlation of 0.35, while the maximum is 0.90. The Age feature is the least correlated with the other features. Most scatter plots and marginal density functions display structures and marginals somewhat similar to the Gaussian distribution, except those related to the S4 feature, which has a multi-modal marginal. We split the data into a training and test data set at a 75–25 ratio, and we let the predictive model be a principle component regression (PCR) model with six principal components. This model outperformed the linear model and cross-validated random forest, XGBoost, CatBoost, PPR, and NN models in prediction error on the test data. The PCR model is not easily interpretable as it does not directly depend on the features but on their principal components.

Table 3 shows that the LM separate, GAM separate, and PPR separate methods obtain the lowest MSE_v scores, with the VAEAC, Gaussian, and copula approaches having nearly as low MSE_v scores. We are not surprised that the latter two methods are competitive due to the Gaussian-like structures in the Diabetes data set. The LM separate method is the fastest approach, with a CPU time of 1.9 seconds.

5.3 Red wine

The Red Wine data set contains information about variants of the Portuguese Vinho Verde wine (Cortez et al. 2009). The response is a `quality` score between 0 and 10, while the $M = 11$ continuous features are based on physicochemical tests: `fixed acidity`, `volatile acidity`, `citric acid`, `residual sugar`, `chlorides`, `free sulfur dioxide`, total `sulfur dioxide`, `density`, `pH`, `sulphates`, and `alcohol`. For the Red Wine data set, most scatter plots and marginal density functions display structures and marginals far from the Gaussian distribution, as most of the marginals are right-skewed. Many of the features have no to moderate correlation, with a mean absolute correlation of 0.20, while the largest correlation in absolute value is 0.683 between `pH` and `fix_acid`. The data set contains 1599 wines, and we split it into a training (1349) and a test (250) data set. A cross-validated XGBoost model and a random forest with 500 trees perform equally well on the test data, and we use the latter as the predictive model f.

Table 3 shows that the `RF separate` approach is the best method by far. Next, we have the `CatBoost separate`, `RF surrogate`, `empirical`, and `VAEAC` methods. The `RF surrogate` and `CatBoost surrogate` perform well compared to the other `surrogate regression` methods. The good performance of the non-smooth `RF separate` and `CatBoost separate` methods on the non-smooth predictive model f supports our findings from the simulation studies, where we observed that using a `separate regression` method with the same form as f was beneficial. The `generative` methods perform better than the `GH` and `copula` methods, while the `Gaussian` method falls behind. This is intuitive as the data distribution of the Red Wine data set is far from the Gaussian distribution.

5.4 Adult

The Adult data set is based on the 1994 Census database, and the goal is to predict whether a person makes over \$50,000 a year based on $M = 14$ mixed features: `age` (cont.), `workclass` (7 cat.), `fnlwgt` (cont.), `education` (16 cat.), `education-num` (cont.), `marital-status` (7 cat.), `occupation` (14 cat.), `relationship` (6 cat.), `race` (5 cat.), `sex` (2 cat.), `capital-gain` (cont.), `capital-loss` (cont.), `hours-per-week` (cont.), and `native-country` (41 cat.). The pairwise Pearson correlation coefficients for the continuous features are all close to zero, with a mean absolute correlation of 0.06. The data set contains 30,162 individuals, and we split it into a training (30,000) and a test (162) data set. We train a CatBoost model on the training data to predict an individual's probability of making over \$50,000 a year and use the test data to compute the evaluation criterion. We used a relatively small test set due to memory constraints, and we chose the CatBoost as it outperformed the other prediction models we fitted (LM, GAM, RF, NN).

Table 3 shows that the best method is the `CatBoost separate` approach, while second place is shared by the `RF separate` and `VAEAC` methods. Note that the difference in the MSE_v score is very small. Like in the previous experiments, we observe that using a `separate regression` method with the same form as f

Table 4 Decomposed computation times for the Adult data set experiment with $M = 14$, $N_{\text{train}} =$30,000, and $N_{\text{test}} = 162$

Method	Training	Generating $\boldsymbol{x}_{\mathcal{S}}^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
`Independence`	0.0	34:31.9	42:14.4	1:16:46.3	0.041
`VAEAC`	1:05:55:35.9	4:05:53:10.4	42:17.4	5:12:31:03.7	0.027
`LM sep.`	8:38:57.2	—	21:16.3	9:00:13.5	0.043
`GAM sep.`	2:37:14.8	—	1:37.8	2:38:52.6	0.033
`PLS sep.`	25:04:43:30.2	—	17:21.1	25:05:00:51.3	0.046
`PPR sep.`	14:12:16:08.6	—	3:39.0	14:12:19:47.6	0.032
`RF sep.`	98:13:17:15.8	—	16:11.6	98:13:33:27.4	0.027
`RF-def sep.`	12:49:17.0	—	8:39.0	12:57:56.0	0.028
`CatBoost sep.`	35:09:59:20.8	—	38.3	35:09:59:59.1	**0.026**
`NN-Frye sur.`	3:16:10:45.4	—	3:04.9	3:16:13:50.3	0.085
`NN-Olsen sur.`	3:11:31:47.6	—	2:10.3	3:11:33:57.9	0.045

is beneficial. The `ctree` approach supports mixed data, but we deemed it infeasible due to a very long computation time. Furthermore, the `surrogate regression` methods based on (5) ran out of memory as $\mathcal{X}_{\text{aug}}$ consists of 30,000 $\times (2^{14} - 2) =$ 491,460,000 training observations.

In Table 4, we decompose the Adult computation times reported in Table 3 in the same manner as in Sect. 4.3. We see that the training times dwarf the predicting times for the regression-based methods, while the opposite holds for the Monte Carlo-based methods when including the generating times. Recall that the training time is a one-time upfront cost. In contrast, the generating and predicting times are more interesting as they express the time needed to compute future Shapley value explanations. In particular, the sum of the generating and predicting times divided by $N_{\text{test}} = 162$ yields an estimate of the time needed to explain one new prediction in the future.

The long training time of the `RF separate` method is due to the hyperparameter tuning, as discussed in Sect. 4.3. In the Supplement, we include a random forest method without hyperparameter tuning but instead with default values and a lower number of trees, denoted by `RF-def separate`. It obtains an $\text{MSE}_v = 0.028$ with a training time of 12:49:17.0, i.e., a 99.5% reduction in the training time. The competitive MSE_v illustrates that hyperparameter tuning was not essential in this experiment, unlike in the simulation studies where hyperparameter tuning was crucial for obtaining a competitive method. Furthermore, as discussed in Sect. 2.2.3, using an approximation strategy with $N_{\mathcal{S}} < 2^M - 2$ coalitions would also speed up the computations. This could also make the other `surrogate regression` methods applicable, as the size of the augmented training data would be reduced by a factor of $N_{\mathcal{S}}/(2^M - 2)$.

6 Recommendations

In this section, we propose a list of advice for when to use the different methods and method classes based on the results of the simulations studies and the real-world data experiments. The list is not exhaustive. Hence, it must not be interpreted as definite rules but as guidance and points that should be considered when using conditional Shapley values for model explanation.

1. For data sets with no or minuscule feature dependencies, the `independence` approach is the simplest method to use.
2. In general, a `parametric` approach with the correct (or nearly correct) parametric assumption about the data distribution generates the most accurate Shapley values.

   The `copula` method does not make an assumption about the marginals of the data, but rather the copula/dependence structure, which makes it a more robust method.
   For features that do not fit the assumed distribution in the `parametric` approach, one can consider transformations, for example, power transformations, to make the data more Gaussian-like distributed.
   For categorical features, one can use, e.g., encodings or entity embeddings to represent the categorical features as numerical. This is needed, as no directly applicable multivariate distribution exists for mixed data. However, there exist copulas that support mixed data.
   If the `parametric` methods are not applicable, the next best option is (often) a `generative` or `separate regression` method, where all considered approaches support mixed data sets by default.

3. For the `separate` and `surrogate regression` methods, using a method with the same form as the predictive model f provides more precise Shapley value estimates.

   For some predictive models, e.g., the linear regression model in Fig. 2, we know that the true conditional model is also a linear model. Thus, using a regression method that can model a linear model (e.g., `lm`, `GAM`, `PPR`) produces more accurate Shapley values. However, the form of the true conditional model is usually unknown for most predictive models.
   It is important that the regression method used is flexible enough to estimate/model the predictive model f properly.
   In the numerical simulation studies, the `separate regression` methods performed relatively better compared to the other method classes for higher feature dependence. In the real-world experiments, the `separate regression` methods were also (among) the best approaches on data sets with moderate dependence.
   In general, conducting hyperparameter tuning of the regression methods improves the precision of the produced explanations, but this increases the computation time.
   In the simulation studies, a `PPR separate` approach with fixed $L = |\mathcal{S}|$ (often) provides fast and accurate Shapley value explanations; see the Supplement.

4. The modeling of the conditional distributions $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$ in the Monte Carlo-based methods is independent of the predictive model f.

For popular data sets, one can fine-tune an `empirical`, `parametric`, or `generative` method and let other researchers reuse the method to estimate Shapley values for their own predictive models.
If a researcher is to explain several predictive models fitted to the same data, then reusing the generated Monte Carlo samples will save computation time.

5. There is a time-accuracy trade-off between the different method classes and approaches.

   The simplest `separate` and `surrogate regression` methods are rapidly trained, while the complex methods are time-consuming. This is, however, a one-time upfront time cost. In return, all regression-based methods produce the Shapley value explanations almost instantly. Thus, developers can develop the predictive model f simultaneously with a suitable regression-based method and deploy them together. The user of f will then get predictions and explanations almost instantaneously.
   In contrast, several of the Monte Carlo-based methods are trained considerably faster than many of the regression-based methods but are, in return, substantially slower at producing the Shapley value explanations. Generating Monte Carlo samples and using them to estimate the Shapley values for new predictions are computationally expensive and cannot be done in the development phase. Thus, the Monte Carlo-based methods cannot produce explanations in real-time.
   If the predictive model f is computationally expensive to call, then the Monte Carlo-based methods will be extra time-consuming due to $\mathcal{O}(KN2^M)$ calls to f. Here, K, N, and M are the number of Monte Carlo samples, predictions to explain, and features, respectively. In contrast, the `separate` and `surrogate regression` methods make only $\mathcal{O}(N2^M)$ calls to their fitted regression model(s).
   The regression-based methods can be computationally tractable when the Monte Carlo-based methods are not, for example, when N is large. We can reduce the time by decreasing the number of Monte Carlo samples K, but this results in less stable and accurate Shapley value explanations.
   If accurate Shapley values are essential, then a suitable `parametric`, `generative`, or `separate regression` approach with the same form as f yields desirable estimates, depending on the dependence level. The `NN-Olsen surrogate` method also provided accurate Shapley values for some real-world data sets. Furthermore, hyperparameter tuning should be conducted for extra accuracy.
   If coarsely estimated Shapley values are acceptable, then some of the simple `separate regression` methods can be trained and produce estimates almost immediately, such as `LM separate`. The `PPR separate` approach with fixed $L = |\mathcal{S}|$ is often a fair trade-off between time and accuracy, especially for smooth predictive functions.

6. The number of training observations N_{train} did not significantly affect the method classes' overall ordering in our simulation studies. However, individual

approaches, such as the `PPR separate`, performed even better when trained on more training observations.

7. All method classes benefit from having access to multiple CPUs when properly implemented. For example, the Monte Carlo-based approaches can generate the samples for different coalitions and test observations on different cores, and the same when predicting the responses. A `separate regression` method can train the individual models in parallel, while a `surrogate regression` method can cross-validate the model's hyperparameters on different cores.
8. For high-dimensional settings, the number of models to fit in the `separate regression` class is infeasible. Then, the `surrogate regression` methods and the `VAEAC` approach with arbitrary conditioning can be useful. However, their accuracy will likely also decrease with higher dimensions. In high-dimensional settings, one can, e.g., group the features into relevant groups (Jullum et al. 2021) or use approximation strategies to simplify the Shapley value computations, as described in Sect. 2.2.3.

7 Conclusion

In this article, we have discussed a large sample of Monte Carlo integration and regression-based methods used to estimate conditional Shapley values for model explanation. In agreement with the literature (Covert et al. 2021; Chen et al. 2022), we have divided the studied methods into six different method classes. For each class, we have given an overview of the idea, reviewed earlier proposed methods within the class, and finally proposed and developed several new approaches for most classes. The existing and novel approaches have been systematically evaluated through a series of simulation studies with increasing complexity, as such evaluation has until now been lacking in the field of conditional Shapley values (Chen et al. 2022). We also conducted several experiments on real-world data sets from the UCI Machine Learning Repository. The ranking of the method classes and approaches differed slightly in the numerical simulation studies and real-world experiments.

The most accurate Shapley value explanations in the simulation studies were generally produced by a `parametric` method with a correctly (or nearly correctly) assumed data distribution. This is intuitive, as making a correct parametric assumption is advantageous throughout statistics. However, the true data distribution is seldom known, e.g., for real-world data sets. In the simulation studies with moderate feature dependence levels, the second-best method class was generally the `generative` class with the `ctree` and `VAEAC` methods, which outperformed the `independence`, `empirical`, `separate regression`, and `surrogate regression` methods. For high feature dependence, the `separate regression` methods improved relative to the other classes, particularly the `PPR separate` method. Using a `separate regression` method with the same form as the predictive model proved beneficial.

In the real-world experiments, the `parametric` methods fell behind the best approaches, except for the simplest data set with Gaussian-like structures. In general, the best approaches in the real-world data set experiments belong to the `separate` regression method class and have the same form as the predictive model. However, the NN-Olsen surrogate method tied the best separate regression method in one experiment, and the VAEAC approach was marginally more precise in another experiment. The second-best method class varied for the different data sets, with all method classes, except the independence and empirical, taking at least one second place each.

In addition to the accuracy of the methods, we also investigated the computation time. The regression-based methods are often slowly trained, but they produce the Shapley value explanations almost instantaneously. In contrast, the Monte Carlo-based methods are often faster to train but drastically slower at generating the Shapley value explanations. Finally, we gave some recommendations and considerations for when to use the different method classes and approaches.

In further work, one direction is to extend the investigation into higher dimensions to verify that the tendencies and order of the methods we discovered remain. However, one would then probably need to sample a subset of the coalitions to cope with the exponential complexity of Shapley values. In agreement with Chen et al. (2022), one can also try to determine robust architectures, training procedures, and hyperparameter optimization for the generative and surrogate regression methods, investigate how non-optimal approaches change the estimated conditional Shapley values, and finally evaluate bias in estimated conditional Shapley values for data with known conditional distributions.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10618-024-01016-z.

Acknowledgements The Norwegian Research Council supported this research through the BigInsight Center for Research-based Innovation, Project Number 237718. Furthermore, we thank the anonymous reviewers for their suggestions and feedback on the manuscript. LHBO thanks the Norwegian Artificial Intelligence Research Consortium (NORA) for the opportunity to participate in the Enrichment Scheme at the Alan Turing Institute in the United Kingdom.

Funding Open access funding provided by University of Oslo (incl Oslo University Hospital).

Appendix

In Appendix A, we describe implementation details for the methods in the main text. While we provide more details about the parametric methods in Appendix B.

Appendix A: implementation details

In this section, we describe implementation details for the methods introduced in Sect. 3. We use the R-package `shapr` (Sellereite and Jullum 2019), version 0.2.0, to compute the Shapley values[3]. The package computes the Shapley values as the solution of a weighted least squares problem (Charnes et al. 1988; Lundberg and Lee 2017; Aas et al. 2021a). More precisely, the solution is given by $\boldsymbol{\phi} = (\boldsymbol{Z}^T \boldsymbol{W} \boldsymbol{Z})^{-1} \boldsymbol{Z}^T \boldsymbol{W} \boldsymbol{v} = \boldsymbol{R}\boldsymbol{v}$. The $\boldsymbol{Z}$ matrix is a $2^M \times (M+1)$ binary matrix where the first column consists of 1s and the remaining columns are the $I(\mathcal{S})$ representations from Sect. 3.6.1 for all $\mathcal{S} \in \mathcal{P}(\mathcal{M})$. The $\boldsymbol{W}$ matrix is a $2^M \times 2^M$ diagonal matrix containing the Shapley kernel weights $k(M, |\mathcal{S}|) = (M-1)/(\binom{M}{|\mathcal{S}|}|\mathcal{S}|(M-|\mathcal{S}|))$, while $\boldsymbol{v}$ is a 2^M-dimensional vector containing the estimated contribution functions $\hat{v}(\mathcal{S})$. The $\mathcal{S}$ in the latter two cases resembles the coalition of the corresponding row in $\boldsymbol{Z}$. The $\boldsymbol{Z}$ and $\boldsymbol{W}$ matrices are independent of the instance to be explained and are computed by the `shapr` package, which sets the infinite Shapley kernel weights $k(M, 0) = k(M, M) = \infty$ to a large constant $C = 10^6$. When explaining N_{test} predictions, we replace $\boldsymbol{v}$ with a $2^M \times N_{\text{test}}$ matrix $\boldsymbol{V}$, where column i contains the estimated contribution functions for instance i.

In Sect. 3.7, we described the time complexity of the three method-specific steps when computing Shapley value explanations. Here, we also address the method-independent complexity of setting up and using the Shapley value explanation framework. The computational complexity is related to computing the Shapley value explanations as the solution of the weighted least squares problem described above. The time complexity of computing $\boldsymbol{R}$ with standard schoolbook computations is $\mathcal{O}(M 2^{2M})$, while it is $\mathcal{O}(M N_{\text{test}} 2^M)$ for the matrix multiplication $\boldsymbol{\phi} = \boldsymbol{R}\boldsymbol{V}$.

The `independence`, `empirical`, `Gaussian`, `copula`, and `ctree` methods are implemented in the `shapr` package, and we use default hyperparameter values. For the other methods, we estimate $\boldsymbol{V}$ and multiply it with $\boldsymbol{R}$ to get the estimated Shapley values. Olsen et al. (2022) implement the `VAEAC` approach as an add-on to the `shapr` package, and we use the default architecture and hyperparameters. In the numerical simulation studies in Sect. 4, we train the `VAEAC` approach for 200 epochs and use the estimated model parameters at the epoch with the lowest validation error, where 25% of the data constitutes the validation data. For the more complex real-world data distributions in Sect. 5, the `VAEAC` approach needs more training epochs to learn to model the data distributions properly. For the Abalone$_{\text{cont}}$, Abalone$_{\text{all}}$, Diabetes, Wine, and Adult data sets, we let the number of epochs be: 10,000, 40,000, 5000, 10,000, and 200, respectively. Other configurations than the default architecture and hyperparameters might reduce the number of needed learning epochs. In the Supplement, we provide `VAEAC` approaches with other numbers of epochs in the numerical simulation studies and the real-world data experiments, respectively.

Throughout the article, if not otherwise specified, we use $K = 250$ Monte Carlo samples in (3) for the Monte Carlo-based methods, which Olsen et al. (2022) found to be a fair trade-off between accuracy and computation time. However, recall that

[3] From version 1.0.0, the `shapr` package simplifies its syntax, provides a Python wrapper, and supports batching of the coalitions to reduce memory consumption and enable parallelizing of the computations.

the `empirical` and `ctree` methods (often) use fewer samples and rather weight them, as described in Sects. 3.2 and 3.4.1, respectively. The implementation of the `independence` approach directly samples the $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ feature values from other observations in the training data; hence, it needs no training, and it supports mixed data.

For the `Burr` and `GH` approaches, we estimate the parameters of the distributions by maximizing the likelihood function using the Nelder-Mead optimization routine (Nelder and Mead 1965), with default parameters in the `optim` function in base R (R Core Team 2020). Tuning the hyperparameters of the optimization algorithm and/or using a more advanced fitting procedure might improve the approaches. We run the optimization procedures until convergence. The number of parameters to estimate in the Burr and GH distribution is $2M + 1$ and $\frac{1}{2}(M + 1)(M + 4)$, respectively. The optimization of the `GH` method relies on good starting values, which we get from the `ghyp` package (Weibel et al. 2022). The `ghyp` package uses a sophisticated multi-cycle, expectation, conditional estimation (MCECM) algorithm to estimate the parameters for another more general parameterization of the GH distribution, which lacks closed-form expressions for the conditional distributions.

For the `separate regression` methods, we tune (some of) the hyperparameters of the different methods using 4-fold cross-validation procedures implemented in the packages, by us, or by using the `caret` package (Kuhn 2022). The `LM separate` approach was fitted using the `lm` function in the `stats` package in base R. We use the `mgcv` package (Wood 2022), with default parameters, to fit the `GAM separate` method. Note that in the `mgcv` package, the smoothing parameters in the penalized regression splines are selected by generalized cross-validation during the fitting procedure. The `PPR separate` method uses the `ppr` function in the `stats` package with default parameters, except the number of terms L, which we determine by cross-validation. The `RF separate` approach is based on the `ranger` package (Wright and Ziegler 2017). We use 500 decision trees and the `caret` package to do cross-validation on `mtry` (3 options), `splitrule` (2 options), and `min.node.size` (2 options), while we use default values for the remaining hyperparameters. Finally, the `CatBoost separate` method uses the `CatBoost` algorithm (Prokhorenkova et al. 2018), which is based on gradient-boosted decision trees, with default parameters (most notably, 1000 trees with depth 6). We employ early stopping of the `CatBoost` method if no improvement of the evaluation metric value was made in 100 iterations. One could employ cross-validation to tune the hyperparameters, but this would increase the computation time drastically as the `CatBoost` algorithm has many hyperparameters. An alternative is to tune only some of them.

For the `surrogate regression` methods, we use the same packages as above and tune the same hyperparameters if not otherwise specified. For the `RF surrogate` method, we reduced the number of trees from 500 to 200 due to high computation time. For the `CatBoost surrogate` approach, we increase the maximum number of trees to 10,000, but we still employ the same early stopping regime.

Originally, Frye et al. (2021) let the masking value be -1, as they only consider positive data, but this is not applicable for unbounded features. We let the value be -5 in the simulations and real-world experiments in Sects. 4 and 5, respectively, which is a value not present in the data sets. For the `NN-Frye surrogate` approach, we use the same fully connected neural network and carry out the same cross-validation as in Frye et al. (2021). That is, `depth` $= 2$, `width` $\in \{128, 256, 512\}$, `batch_size` $= 256$, and `learning_rate` $\in \{0.001, 0.0001\}$ in the Adam optimizer (Kingma and Ba 2015). We use 75% of the data to train the networks and the remaining observations as validation data. We use the network parameters at the epoch with the lowest validation error as the final model. Frye et al. (2021) use 2000–10,000 training epochs, while we use `num_epochs` $= 3000$ to make the method more time-wise competitive and as the validation error obtains its minimum long before the last epoch in the simulation studies. Another alternative is to let `num_epochs` be arbitrarily large and stop the training if no improvement has been made to the validation error for a fixed number of epochs, that is, employing early stopping.

In the `NN-Olsen surrogate` approach, we use batch normalization layers, ELU activation functions, and skip connections with summation over each layer in the network. We carry out similar hyperparameter tuning as the `NN-Frye surrogate` approach. That is, `depth` $= 3$, `width` $\in \{32, 64, 128\}$, `num_epochs` $= 500$, `batch_size` $= 128$ (as we duplicate the batch size), and `learning_rate` $\in \{0.01, 0.001, 0.0001\}$ in the Adam optimizer. Instead of specifying `num_epochs`, another option could have been to train the network until a stopping criterion was met, e.g., no improvement in the validation measure for a specific number of epochs. We observe relatively small differences between the nine different hyperparameter choices, and one could thus potentially reduce the training time by a factor of nine by using `width` $= 64$ and `learning_rate` $= 0.001$ as default values. The same also applies to the `NN-Frye surrogate` approach. The networks are implemented in `torch` (Falbel and Luraschi 2022).

In the real-world data experiments in Sect. 5, we omit the cross-validation of the hyperparameters in the `NN surrogate` approaches to make them more time-wise competitive. We let `lr` $= 0.001$ and `width` $= 256$ in the `NN-Frye surrogate` approach, while we use the same learning rate for the `NN-Olsen surrogate` method but we let `width` $= 64$. The convergence rates of the networks' validation errors vary in the different real-world data experiments. Hence, we use different `num_epochs` for each experiment. For the Abalone$_{\text{cont}}$, Abalone$_{\text{all}}$, Diabetes, Wine, and Adult data sets, we let `num_epochs` in the `NN-Frye surrogate` method be: 40,000, 40,000, 10,000, 40,000, and 3000, respectively. The corresponding values for the `NN-Olsen surrogate` method are 20,000, 10,000, 2500, 10,000, and 500. Other configurations than the architecture and hyperparameters set above might reduce the number of needed learning epochs. In the Supplement, we provide `NN surrogate` methods with other numbers of epochs, as a higher `num_epochs` can make the methods more precise but at the cost of increased training time.

Appendix B: additional information about the parametric methods

In this section, we elaborate on the `copula` approach and give a short introduction to the multivariate Burr and generalized hyperbolic distributions.

B.1: Copulas

The definition of an M-dimensional copula is a multivariate distribution, C, with uniformly distributed marginals $\mathcal{U}[0, 1]$. Sklar's theorem states that every multivariate distribution F with marginals $F_1, F_2, \dots, F_M$ can be written as $F(x_1, \dots, x_M) = C(F_1(x_1), \dots, F_M(x_M))$, for some appropriate M-dimensional copula C. In fact, the copula from the previous equation has the expression $C(u_1, \dots, u_M) = F(F_1^{-1}(u_1), \dots, F_M^{-1}(u_M))$, where the $F_j^{-1}(u_j)$s are the inverse distribution functions of the marginals. While other copulas may be used, the Gaussian copula has the benefit that we may use the analytical expressions for the conditionals of the Gaussian distribution.

The Gaussian copula model used by Aas et al. (2021a) is very flexible with regard to the marginal distributions but quite restrictive in the dependence structures it can capture. It can only represent radially symmetric dependence relationships and does not allow for tail dependence (i.e., the joint occurrence of extreme events has a small probability). One can use other copulas in the `copula` approach instead. For example, Aas et al. (2021b) use vine copulas, more specifically, a particular type of R-vines (regular vines) called D-vines (Kurowicka and Cooke 2005) when they estimate conditional Shapley values. Regular vines do not exclude categorical data, but the methods become more complicated when categorical features are included; hence, Aas et al. (2021b) exclude them. Zhao and Udell (2020) propose a semi-parametric algorithm to impute missing values for mixed data sets via a Gaussian copula.

B.2: Burr distribution

The Burr distribution allows for heavy-tailed, skewed marginals, and nonlinear dependencies, which can be found in real-world data sets (Takahasi 1965). The density of the M-dimensional Burr distribution is given by

$$p(\boldsymbol{x}) = \frac{\Gamma(\kappa + M)}{\Gamma(\kappa)} \left(\prod_{m=1}^{M} b_m r_m\right) \frac{\prod_{m=1}^{M} x_m^{b_m - 1}}{\left(1 + \sum_{m=1}^{M} r_m x_m^{b_m}\right)^{\kappa + M}},$$

for $x_m > 0$. The M-dimensional Burr distribution has $2M + 1$ parameters, namely, κ, $b_1, \dots b_M$, and $r_1, \dots, r_M$. Furthermore, the Burr distribution is a compound Weibull distribution with the gamma distribution as compounder (Takahasi 1965), and it can also be seen as a special case of the Pareto IV distribution (Yari and Jafari 2006).

Any conditional distribution of the Burr distribution is in itself a Burr distribution (Takahasi 1965). Without loss of generality, assume that the first $S < M$ features are the unobserved features, then the conditional density $p(x_1, \dots, x_S | x_{S+1} = x_{S+1}^*, \dots, x_M = x_M^*)$, where $\boldsymbol{x}^*$ indicates the conditional values, is an S-dimensional Burr density. The associated parameters are then $\tilde{\kappa}, \tilde{b}_1, \dots, \tilde{b}_S$, and $\tilde{r}_1, \dots, \tilde{r}_S$, where $\tilde{\kappa} = \kappa + M - S$, while $\tilde{b}_j = b_j$ and $\tilde{r}_j = \frac{r_j}{1 + \sum_{m=S+1}^{M} r_m (x_m^*)^{b_m}}$, for all $j = 1, 2, \dots, S$.

B.3: generalized hyperbolic distribution

The generalized hyperbolic distribution $\mathrm{GH}(\lambda, \omega, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\beta})$ is parameterized by an index parameter λ, concentration parameter ω, location vector $\boldsymbol{\mu}$, dispersion matrix $\boldsymbol{\Sigma}$, and skewness vector $\boldsymbol{\beta}$ (Browne and McNicholas 2015). A random variable $\boldsymbol{x}$ is GH distributed if it can be represented by $\boldsymbol{x} = \boldsymbol{\mu} + W\boldsymbol{\beta} + \sqrt{W}\boldsymbol{U}$, where $W \sim \mathrm{GIG}(\lambda, \omega, \omega)$, $\boldsymbol{U} \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ and W is independent of $\boldsymbol{U}$. GIG is the generalized inverse Gaussian distribution introduced by Good (1953). The density of the M-dimensional GH is given by

$$p(\boldsymbol{x}) = \left[\frac{\omega + \delta(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\omega + \boldsymbol{\beta}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}}\right]^{\frac{\lambda - M/2}{2}} \frac{K_{\lambda - M/2}\left(\sqrt{(\omega + \delta(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}))(\omega + \boldsymbol{\beta}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta})}\right)}{(2\pi)^{M/2}|\boldsymbol{\Sigma}|^{1/2}K_\lambda(\omega)\exp\left\{-(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}\right\}},$$

where $\delta(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$ is the squared Mahalanobis distance between $\boldsymbol{x}$ and $\boldsymbol{\mu}$, K_λ is the modified Bessel function of the third kind with index λ.

Wei et al. (2019) showed that when $\boldsymbol{x}$ is partitioned as $(\boldsymbol{x}_{\mathcal{S}}, \boldsymbol{x}_{\bar{\mathcal{S}}})$, the conditional distribution $\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^* \sim \mathrm{GH}^*(\lambda_{\bar{\mathcal{S}}|\mathcal{S}}, \chi_{\bar{\mathcal{S}}|\mathcal{S}}, \phi_{\bar{\mathcal{S}}|\mathcal{S}}, \boldsymbol{\mu}_{\bar{\mathcal{S}}|\mathcal{S}}, \boldsymbol{\Sigma}_{\bar{\mathcal{S}}|\mathcal{S}}, \boldsymbol{\beta}_{\bar{\mathcal{S}}|\mathcal{S}})$, where $\lambda_{\bar{\mathcal{S}}|\mathcal{S}} = \lambda - |\mathcal{S}|/2$, $\chi_{\bar{\mathcal{S}}|\mathcal{S}} = \omega + (\boldsymbol{x}_{\mathcal{S}}^* - \boldsymbol{\mu}_{\mathcal{S}})^T\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}(\boldsymbol{x}_{\mathcal{S}}^* - \boldsymbol{\mu}_{\mathcal{S}})$, $\psi_{\bar{\mathcal{S}}|\mathcal{S}} = \omega + \boldsymbol{\beta}_{\mathcal{S}}^T\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^T\boldsymbol{\beta}_{\mathcal{S}}$, $\boldsymbol{\mu}_{\bar{\mathcal{S}}|\mathcal{S}} = \boldsymbol{\mu}_{\bar{\mathcal{S}}} + \boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}}^T\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}(\boldsymbol{x}_{\mathcal{S}}^* - \boldsymbol{\mu}_{\mathcal{S}})$, $\boldsymbol{\Sigma}_{\bar{\mathcal{S}}|\mathcal{S}} = \boldsymbol{\Sigma}_{\bar{\mathcal{S}}\bar{\mathcal{S}}} - \boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}}^T\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}\boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}}$, and $\boldsymbol{\beta}_{\bar{\mathcal{S}}|\mathcal{S}} = \boldsymbol{\beta}_{\bar{\mathcal{S}}} - \boldsymbol{\Sigma}_{\mathcal{S}\bar{\mathcal{S}}}^T\boldsymbol{\Sigma}_{\mathcal{S}\mathcal{S}}^{-1}\boldsymbol{\beta}_{\mathcal{S}}$. Here, the GH^* indicates that another parameterization of the GH distribution, proposed by McNeil et al. (2015), is used for the conditional distribution due to technical reasons:

$$p(\boldsymbol{x}) = \left[\frac{\chi + \delta(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\psi + \boldsymbol{\beta}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}}\right]^{\frac{\lambda - M/2}{2}} \times \frac{(\psi/\chi)^{\lambda/2}K_{\lambda - M/2}\left(\sqrt{(\chi + \delta(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}))(\psi + \boldsymbol{\beta}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta})}\right)}{(2\pi)^{M/2}|\boldsymbol{\Sigma}|^{1/2}K_\lambda(\sqrt{\chi\psi})\exp\left\{-(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}\right\}}.$$

References

Aas K, Jullum M, Løland A (2021) Explaining individual predictions when features are dependent: more accurate approximations to Shapley values. Artif Intell 298(103):502

Aas K, Nagler T, Jullum M et al (2021) Explaining predictive models using Shapley values and non-parametric vine copulas. Depend Model 9(1):62–81

Adadi A, Berrada M (2018) Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE Access 6:52138–52160

Ancona M, Oztireli C, Gross M (2019) Explaining deep neural networks with a polynomial time algorithm for Shapley value approximation. In: International conference on machine learning. PMLR, pp 272–281

Barndorff-Nielsen O (1977) Exponentially decreasing distributions for the logarithm of particle size. Proc R Soc Lond A Math Phys Sci 353(1674):401–419

Blesch K, Wright MN, Watson D (2023) Unfooling SHAP and SAGE: knockoff imputation for Shapley values. In: World conference on explainable artificial intelligence. Springer, pp 131–146

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Browne RP, McNicholas PD (2015) A mixture of generalized hyperbolic distributions. Can J Stat 43(2):176–198

Charnes A, Golany B, Keane M, et al. (1988) Extremal principle solutions of games in characteristic function form: core, chebychev and shapley value generalizations. In: Sengupta JK, Kadekodi GK (eds) Econometrics of planning and efficiency. Springer, p 123–133

Chen H, Janizek JD, Lundberg S et al (2020) True to the model or true to the data? arXiv preprint arXiv:2006.16234

Chen H, Covert IC, Lundberg SM et al (2022) Algorithms to estimate Shapley value feature attributions. arXiv preprint arXiv:2207.07605

Cortez P, Teixeira J, Cerdeira A et al (2009) Using data mining for wine quality assessment. In: Discovery science: 12th international conference, DS 2009, Porto, Portugal, October 3–5, 2009 12. Springer, pp 66–79

Covert I, Lee SI (2021) Improving KernelSHAP: practical Shapley value estimation using linear regression. In: International conference on artificial intelligence and statistics. PMLR, pp 3457–3465

Covert I, Lundberg SM, Lee SI (2020) Understanding global feature contributions with additive importance measures. Adv Neural Inf Process Syst 33:17212–17223

Covert I, Lundberg S, Lee SI (2021) Explaining by removing: a unified framework for model explanation. J Mach Learn Res 22(209):1–90

Deng X, Papadimitriou CH (1994) On the complexity of cooperative solution concepts. Math Oper Res 19(2):257–266

Efron B, Hastie T, Johnstone I et al (2004) Least angle regression. Ann Stat 32(2):407–451

European Commission (2016) Regulation EU 2016/679 of the European parliament and of the council of 27 April 2016; general data protection regulation. Official Journal of the European Union

Faigle U, Kern W (1992) The Shapley value for cooperative games under precedence constraints. Int J Game Theory 21:249–266

Falbel D, Luraschi J (2022) torch: Tensors and neural networks with 'GPU' acceleration. https://CRAN.R-project.org/package=torch, r package version 0.9.0

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139

Friedman JH, Stuetzle W (1981) Projection pursuit regression. J Am Stat Assoc 76(376):817–823

Frye C, de Mijolla D, Begley T et al (2021) Shapley explainability on the data manifold. In: International conference on learning representations

Fryer D, Strümke I, Nguyen H (2021) Shapley values for feature selection: the good, the bad, and the axioms. arXiv preprint arXiv:2102.10936

Giudici P, Raffinetti E (2021) Shapley–Lorenz explainable artificial intelligence. Expert Syst Appl 167(114):104

Good IJ (1953) The population frequencies of species and the estimation of population parameters. Biometrika 40(3–4):237–264

Gower JC (1971) A general coefficient of similarity and some of its properties. Biometrics 27:857–871

Guo C, Berkhahn F (2016) Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737

Guo M, Zhang Q, Liao X et al (2019) An interpretable machine learning framework for modelling human decision behavior. arXiv:1906.01233

Hastie T, Tibshirani R, Friedman JH et al (2009) The elements of statistical learning: data mining, inference, and prediction, vol 2. Springer, Cham

Ho TK (1995) Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. IEEE, pp 278–282

Hothorn T, Hornik K, Zeileis A (2006) Unbiased recursive partitioning: a conditional inference framework. J Comput Graph Stat 15(3):651–674

Ivanov O, Figurnov M, Vetrov D (2019) Variational autoencoder with arbitrary conditioning. In: International conference on learning representations

Jethani N, Sudarshan M, Covert IC et al (2021) FastSHAP: real-time Shapley value estimation. In: International conference on learning representations

Johansson U, Sönströd C, Norinder U et al (2011) Trade-off between accuracy and interpretability for predictive in silico modeling. Future Med Chem 3(6):647–663

Jullum M, Løland A, Huseby RB et al (2020) Detecting money laundering transactions with machine learning. J Money Laund Control 23(1):173–186

Jullum M, Redelmeier A, Aas K (2021) Efficient and simple prediction explanations with groupShapley: a practical perspective. In: Musto C, Guidotti R, Monreale A et al (eds) Italian workshop on explainable artificial intelligence 2021. XAI.it, pp 28–43. http://ceur-ws.org/Vol-3014/paper3.pdf
Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: 3rd international conference on learning representations
Kingma DP, Welling M (2014) Auto-encoding variational Bayes. In: 2nd international conference on learning representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, conference track proceedings
Kingma DP, Welling M (2019) An introduction to variational autoencoders. Found Trends Mach Learn 12:307–392
Kourou K, Exarchos TP, Exarchos KP et al (2015) Machine learning applications in cancer prognosis and prediction. Comput Struct Biotechnol J 13:8–17
Kuhn M (2022) caret: classification and Regression Training. https://CRAN.R-project.org/package=caret, r package version 6.0-93
Kumar IE, Venkatasubramanian S, Scheidegger C et al (2020) Problems with Shapley-value-based explanations as feature importance measures. arXiv:2002.11097
Kurowicka D, Cooke R (2005) Distribution-free continuous Bayesian belief. Mod Stat Math Methods Reliab 10:309
Kvamme H, Sellereite N, Aas K et al (2018) Predicting mortgage default using convolutional neural networks. Expert Syst Appl 102:207–217
Lansford JL, Barnes BC, Rice BM et al (2022) Building chemical property models for energetic materials from small datasets using a transfer learning approach. J Chem Inf Model 62(22):5397–5410
Lipovetsky S, Conklin M (2001) Analysis of regression in game theory approach. Appl Stoch Model Bus Ind 17(4):319–330
Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. In: Advances in neural information processing systems, pp 4765–4774
Lundberg SM, Erion GG, Lee SI (2018) Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888
Lundberg SM, Erion G, Chen H et al (2020) From local explanations to global understanding with explainable AI for trees. Nat Mach Intell 2(1):56–67
Luo Y, Tseng HH, Cui S et al (2019) Balancing accuracy and interpretability of machine learning approaches for radiation treatment outcomes modeling. BJR| Open 1(1):20190021
Mase M, Owen AB, Seiler B (2019) Explaining black box decisions by Shapley cohort refinement. arXiv preprint arXiv:1911.00467
Mayr A, Binder H, Gefeller O et al (2014) The evolution of boosting algorithms. Methods Inf Med 53(06):419–427
McNeil AJ, Frey R, Embrechts P (2015) Quantitative risk management: concepts, techniques and tools-revised edition. Princeton University Press, Princeton
Merrick L, Taly A (2020) The explanation game: explaining machine learning models using Shapley values. Machine learning and knowledge extraction. Lecture Notes in Computer Science. Springer International Publishing, Cham, pp 17–38. https://doi.org/10.1007/978-3-030-57321-8_2
Mitchell R, Cooper J, Frank E et al (2022) Sampling permutations for Shapley value estimation. J Mach Learn Res 23(1):2082–2127
Molnar C (2022) Interpretable machine learning, 2nd edn. https://christophm.github.io/interpretable-ml-book
Molnar C (2023) Interpreting machine learning models with SHAP, 1st edn. https://christophmolnar.com/books/shap/
Nash WJ, Sellers TL, Talbot SR et al (1994) The population biology of abalone (*Haliotis* species) in Tasmania. I. Blacklip abalone (*H. rubra*) from the North Coast and the Islands of Bass Strait sea fisheries division, technical report 48, p 411
Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7(4):308–313
Okhrati R, Lipani A (2021) A multilinear sampling algorithm to estimate Shapley values. In: 2020 25th international conference on pattern recognition (ICPR). IEEE, pp 7992–7999
Olsen LHB (2023) Precision of individual shapley value explanations. arXiv preprint arXiv:2312.03485
Olsen LHB, Glad IK, Jullum M et al (2022) Using Shapley values and variational autoencoders to explain predictive models with dependent mixed features. J Mach Learn Res 23(213):1–51
Owen AB (2014) Sobol' indices and Shapley value. SIAM/ASA J Uncertain Quantif 2(1):245–251

Podani J (1999) Extending Gower's general coefficient of similarity to ordinal characters. Taxon 48(2):331–340

Prokhorenkova L, Gusev G, Vorobev A et al (2018) CatBoost: unbiased boosting with categorical features. In: Bengio S, Wallach H, Larochelle H et al (eds) Advances in neural information processing systems, vol 31. Curran Associates Inc., Red Hook

R Core Team (2020) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Redelmeier A, Jullum M, Aas K (2020) Explaining predictive models with mixed features using Shapley values and conditional inference trees. In: International cross-domain conference for machine learning and knowledge extraction. Springer, pp 117–137

Rezende DJ, Mohamed S, Wierstra D (2014) Stochastic backpropagation and approximate inference in deep generative models. In: International conference on machine learning. PMLR, pp 1278–1286

Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215

Sellereite N, Jullum M (2019) shapr: an r-package for explaining machine learning models with dependence-aware Shapley values. J Open Source Softw 5(46):2027

Shapley LS (1953) A value for n-person games. Contrib Theory Games 2(28):307–317

Strumbelj E, Kononenko I (2010) An efficient explanation of individual classifications using game theory. J Mach Learn Res 11:1–18

Strumbelj E, Kononenko I (2014) Explaining prediction models and individual predictions with feature contributions. Knowl Inf Syst 41(3):647–665

Strumbelj E, Kononenko I, Sikonja MR (2009) Explaining instance classifications with interactions of subsets of feature values. Data Knowl Eng 68(10):886–904

Sundararajan M, Najmi A (2020) The many Shapley values for model explanation. In: International conference on machine learning. PMLR, pp 9269–9278

Takahasi K (1965) Note on the multivariate Burr's distribution. Ann Inst Stat Math 17(1):257–260

Vilone G, Rizzo L, Longo L (2020) A comparative analysis of rule-based, model-agnostic methods for explainable artificial intelligence. In: Proceedings for the 28th AIAI Irish conference on artificial intelligence and cognitive science, Dublin, Ireland, December 7–8. Technological University Dublin, pp 85–96

Wang R, Wang X, Inouye DI (2020) Shapley explanation networks. In: International conference on learning representations

Wei Y, Tang Y, McNicholas PD (2019) Mixtures of generalized hyperbolic distributions and mixtures of skew-t distributions for model-based clustering with incomplete data. Comput Stat I Data Anal 130:18–41

Weibel M, Luethi D, Breymann W (2022) ghyp: generalized Hyperbolic distribution and its special cases. https://CRAN.R-project.org/package=ghyp, r package version 1.6.2

Williamson B, Feng J (2020) Efficient nonparametric statistical inference on population feature importance using Shapley values. In: International conference on machine learning. PMLR, pp 10282–10291

Wood S (2006) Low-rank scale-invariant tensor product smooths for generalized additive mixed models. Biometrics 62(4):1025–1036

Wood S (2022) mgcv: mixed GAM computation vehicle with automatic smoothness estimation. https://CRAN.R-project.org/package=mgcv, r package version 1.8.40

Wood SN (2006) Generalized additive models: an introduction with R. Chapman and Hall/CRC, Boca Raton

Wright MN, Ziegler A (2017) ranger: a fast implementation Dof random forests for high dimensional data in C++ and R. J Stat Softw 77(1):1–17. https://doi.org/10.18637/jss.v077.i01

Yari G, Jafari AM (2006) Information and covariance matrices for multivariate Pareto (iv), Burr, and related distributions. Int J Ind Eng Prod Res 17:61–69

Zhao Y, Udell M (2020) Missing value imputation for mixed data via Gaussian copula. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 636–646