Journal of Machine Learning Research 23 (2022) 1-51 Submitted 11/21; Revised 7/22; Published 7/22

Using Shapley Values and Variational Autoencoders to Explain Predictive Models with Dependent Mixed Features

Lars Henry Berge Olsen LHOLSEN@MATH.UIO.NO
Ingrid Kristine Glad GLAD@MATH.UIO.NO
Department of Mathematics
University of Oslo
Moltke Moes vei 35, Niels Henrik Abels hus, 0851 Oslo, Norway

Martin Jullum JULLUM@NR.NO
Kjersti Aas KJERSTI@NR.NO
Norwegian Computing Center
Gaustadalleen 23a, Kristen Nygaards hus, 0373 Oslo, Norway

Editor: Pradeep Ravikumar

Abstract

Shapley values are today extensively used as a model-agnostic explanation framework to explain complex predictive machine learning models. Shapley values have desirable theoretical properties and a sound mathematical foundation in the field of cooperative game theory. Precise Shapley value estimates for dependent data rely on accurate modeling of the dependencies between all feature combinations. In this paper, we use a variational autoencoder with arbitrary conditioning (`VAEAC`) to model all feature dependencies simultaneously. We demonstrate through comprehensive simulation studies that our `VAEAC` approach to Shapley value estimation outperforms the state-of-the-art methods for a wide range of settings for both continuous and mixed dependent features. For high-dimensional settings, our `VAEAC` approach with a non-uniform masking scheme significantly outperforms competing methods. Finally, we apply our `VAEAC` approach to estimate Shapley value explanations for the Abalone data set from the UCI Machine Learning Repository.

Keywords: Explainable artificial intelligence, prediction explanation, Shapley values, variational autoencoder, feature dependence

1. Introduction

Explainable artificial intelligence (XAI) and interpretable machine learning (IML) have become active research fields in recent years (Adadi and Berrada, 2018; Molnar, 2019; Covert et al., 2021). This is a natural consequence as complex machine learning (ML) models are now applied to solve supervised learning problems in many high-risk areas: cancer prognosis (Kourou et al., 2015), credit scoring (Kvamme et al., 2018), and money laundering detection (Jullum et al., 2020). The high prediction accuracy of complex ML models may come at the expense of model interpretability, as discussed by Johansson et al. (2011); Guo et al. (2019); Luo et al. (2019), while Rudin (2019) conjectures that equally accurate but interpretable models exist across domains even though they might be hard to find. As the goal of science is to gain knowledge from the collected data, the use of black-

box models hinders the understanding of the underlying relationship between the features and the response, and thereby curtails scientific discoveries.

Model explanation frameworks from the XAI field extract the hidden knowledge about the underlying data structure captured by a black-box model, making the model's decision-making process transparent. This is crucial for, e.g., medical researchers who apply an ML model to obtain well-performing predictions but who simultaneously also strive to discover important risk factors. Another driving factor is the *Right to Explanation* legislation in the European Union's General Data Protection Regulation (GDPR) which gives a data subject the right to obtain an explanation of decisions reached by algorithms that significantly affect the individual (European Commission, 2016).

A promising explanation methodology, with a strong mathematical foundation and unique theoretical properties within the field of cooperative game theory, is *Shapley values* (Shapley, 1953). Within XAI, Shapley values are most commonly used as a *model-agnostic* explanation framework for individual predictions, i.e., *local explanations*. The methodology has also been used to provide *global explanations*, see Owen (2014); Covert et al. (2020); Frye et al. (2021); Giudici and Raffinetti (2021). Model-agnostic means that Shapley values do not rely on model internals and can be used to compare and explain different ML models trained on the same supervised learning problem. Examples of ML models are neural networks (Gurney, 2018), random forest (Breiman, 2001), and XGBoost (Chen et al., 2015). Local explanation means that Shapley values do not explain the global model behavior across all data instances but rather locally for an individual observation. See Molnar (2019) for an overview and detailed introduction to other explanation frameworks.

Shapley values originated in cooperative game theory but have been reintroduced as a framework for model explanation by Strumbelj and Kononenko (2010, 2014); Lundberg and Lee (2017); Lundberg et al. (2018). Originally, Shapley values described a possible solution concept of how to fairly distribute the payout of a game to the players based on their contribution to the overall cooperation/payout. The solution concept is based on several desirable axioms, for which Shapley values are the unique solution. When applying Shapley values as an explanation framework, we treat the features as the "players", the machine learning model as the "game", and the corresponding prediction as the "payout". In this framework, the difference between the prediction of a specified individual and the global average prediction is fairly distributed among the associated features.

Lundberg and Lee (2017) were a major driving force in the popularization of Shapley values for model explanation with their Python library `SHAP`. In their approach, they implicitly assumed independence between the features, as this partially simplifies the Shapley value estimation. However, the estimation is still fundamentally computationally expensive as the number of terms involved in the Shapley value formula, elaborated in Section 2, grows exponentially with the number of features. Furthermore, in observational studies, the features are rarely statistically independent and the `independence` approach can therefore compute incorrect explanations, see, for example, Merrick and Taly (2020); Frye et al. (2021); Aas et al. (2021a).

Aas et al. (2021a) extend the ideas of Lundberg and Lee (2017) and propose three methods for modeling the dependence between the features: the `Gaussian`, `copula`, and `empirical` approaches. The same paper demonstrates through simulation studies that these conditional approaches obtain more precise Shapley value estimates, in terms of the

mean absolute error, than the `independence` approach for moderate to strong dependence between some or all of the features. Aas et al. (2021b) further improve the explanation precision by using non-parametric vine copulas (Joe, 1996). Redelmeier et al. (2020) extend the methodology to mixed data, that is, continuous and categorical features, by modeling the feature dependencies using conditional inference trees (Hothorn et al., 2006). However, these improvements further increase the computational complexity of Shapley values, as we in addition need to estimate the feature dependencies between all feature combinations. Jullum et al. (2021) propose to explain groups of related features instead of individual features to decrease the fundamental computational problem for Shapley values.

The above papers use methods from the statistical community to model the feature dependencies, but machine learning methods can also be applied. Two promising ML methodologies for constructing generative models for conditional distributions are *generative adversarial network* (`GAN`) (Goodfellow et al., 2014) and *variational autoencoder* (`VAE`) (Kingma and Welling, 2014; Rezende et al., 2014). Both the `VAE` and `GAN` frameworks consist of neural networks (NNs) but with different model setups. They have been used in a wide variety of generative modeling tasks; human image synthesis (Karras et al., 2020), simulating gravitational lensing for dark matter research (Mustafa et al., 2019), blending and exploring musical scores (Roberts et al., 2018; Simon et al., 2018), and generating coherent novel sentences (Bowman et al., 2016).

The `VAE` can be seen as an extension of the expectation-maximization (EM) algorithm (Kingma and Welling, 2014, 2019; Ding, 2021). They optimize the same objective function, but only the `VAE` works if the expectations are intractable, as it uses stochastic gradient ascent on an unbiased estimator of the objective function. The `VAE` is a probabilistic extension of the traditional autoencoder (AE) (Bengio, 2009), and the AE with linear activation functions will learn principal components like vanilla PCA, a well-known special case of the AE (Bourlard and Kamp, 1988). Rezende et al. (2014) mention in passing that a special case of the `VAE` reduces to factor analysis (Bartholomew et al., 2011), a cousin of probabilistic PCA (Tipping and Bishop, 1999), while Dai et al. (2018) relate the `VAE` to robust PCA (Candès et al., 2011).

Both frameworks have methods that could potentially be used to estimate Shapley values, namely *generative adversarial imputation nets* (`GAIN`) (Yoon et al., 2018) and *conditional variational autoencoders* (`CVAE`) (Sohn et al., 2015), respectively. In this paper, we use the *variational autoencoder with arbitrary conditioning* (`VAEAC`) by Ivanov et al. (2019), which is a generalization of `CVAE`. `VAEAC` estimates the dependencies between all features simultaneously with a *single* variational autoencoder and it supports mixed data.

That `VAEAC` uses a single model is a great advantage compared to the methods of Redelmeier et al. (2020); Aas et al. (2021a,b), which fit a new model for each feature combination. As the number of feature combinations grows exponentially with the number of features, the `VAEAC` methodology can reduce the computational burden of Shapley values as a model explanation framework. An abstract visualisation of how the `VAEAC` approach estimates feature dependencies is presented in Figure 1. The figure illustrates how `VAEAC` uses two probabilistic *encoders* to learn latent representations of the data and a single probabilistic *decoder* to map the latent representation back to the feature space while taking the feature dependencies into account.

In this article, we focus on how we can use the `VAEAC` methodology to compute *local* Shapley values in a regression setting. A related approach has been used by Frye et al. (2021) for obtaining *global* explanations by aggregating local Shapley values in a classification setting. The degree of similarity with our `VAEAC` approach to Shapley value estimation is not straightforward to quantify, as they provide little details on their method. There are at least three main differences as far as we are concerned. First, we use a novel non-uniform *masking scheme* which greatly increases the performance of the `VAEAC` methodology in high-dimensional simulation studies. Second, our approach handles *missing feature values*, and third, it is resilient to *vanishing gradients* as we use skip-layer connections and LeakyReLU. Additionally, we conduct thorough simulation studies where our `VAEAC` approach to Shapley value estimation outperforms the current state-of-the-art methods (Lundberg and Lee, 2017; Redelmeier et al., 2020; Aas et al., 2021a) implemented in the R-package `shapr` (Sellereite and Jullum, 2019).

The rest of the paper is organized as follows. In Section 2, we introduce the concept of Shapley values in the game-theoretical setting and as a local model-agnostic explanation framework with individual explanations. The variational autoencoder with arbitrary conditioning (`VAEAC`) is introduced in Section 3, where we also show how the methodology is used to estimate precise Shapley values for dependent features. In Section 4, we illustrate the excellent performance of our `VAEAC` approach compared to the state-of-the-art methods through exhaustive simulations studies. Then, in Section 5, we use our `VAEAC` approach to generate Shapley value explanations for the Abalone data set (Nash et al., 1994) from the UCI Machine Learning Repository. Section 6 ends the paper with some concluding remarks and potential further work. In the Appendix, we provide full mathematical derivations, implementations details, and additional simulation studies.

2. Shapley Values

In this section, we first describe Shapley values in cooperative game theory before we discuss how they can be used for 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 $v(\mathcal{S})$ for the *contribution function* 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 satisfying 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 $\{v_1, \ldots, v_n\}$, 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

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

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 $\mathcal{S}$. The proposed payouts, that is, the Shapley values, are said to be *fair* as they satisfy the four axioms, which are discussed in more detail in, for example, Shapley (1953); Roth (1988); Hart (1989).

2.2 Shapley Values in Model Explanation

Assume that we are in a supervised learning setting where we want to explain a predictive model $f(\boldsymbol{x})$ trained on $\{\boldsymbol{x}^{[i]}, y^{[i]}\}_{i=1,\ldots,N_{\text{train}}}$, 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.

Shapley values as a model-agnostic explanation framework (Strumbelj and Kononenko, 2010, 2014; Lundberg and Lee, 2017) enable us to fairly explain the prediction $\hat{y} = f(\boldsymbol{x})$ for a specific feature vector $\boldsymbol{x} = \boldsymbol{x}^*$. 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 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 outcome of $f(\boldsymbol{x})$ conditioned on the features in $\mathcal{S}$ taking on the values $\boldsymbol{x}_{\mathcal{S}}^*$. That is,

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

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, $\bar{\mathcal{S}} = \mathcal{M} \setminus \mathcal{S}$, 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 summarises 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. Note that the last equality of (2) only holds for continuous features. That is, the integral should be replaced by sums for discrete or categorical features in $\boldsymbol{x}_{\bar{\mathcal{S}}}$, if there are any, and $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ is then no longer continuous.

The contribution function in (2) is also used by, for example, Chen et al. (2020); Covert et al. (2020); Redelmeier et al. (2020); Frye et al. (2021); Aas et al. (2021a,b). Covert et al.

(2021) argue that the conditional approach in (2) is the only approach that is consistent with standard probability axioms. In practice, the contribution function $v(\mathcal{S})$ needs to be empirically approximated by, e.g., Monte Carlo integration for all $\mathcal{S} \in \mathcal{P}(\mathcal{M})$. That is,

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

where f is the machine learning model, $\boldsymbol{x}^{(k)}_{\bar{\mathcal{S}}} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$, for $k = 1, 2, \ldots, K$, and K is the number of Monte Carlo samples.

Lundberg and Lee (2017) assume feature independence, i.e., $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}) = p(\boldsymbol{x}_{\bar{\mathcal{S}}})$, which means that $\boldsymbol{x}^{(k)}_{\bar{\mathcal{S}}}$ can be randomly sampled from the training set. However, in observational studies, the features are rarely statistically independent. Thus, the `independence` approach may lead to incorrect explanations for real data, as discussed in, for example, Merrick and Taly (2020); Aas et al. (2021a); Frye et al. (2021). Redelmeier et al. (2020); Aas et al. (2021a,b) use different methods from the statistical community to model the feature dependencies and generate the conditional samples $\boldsymbol{x}^{(k)}_{\bar{\mathcal{S}}} \sim p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$.

The aforementioned approaches need to estimate the conditional distributions for all feature combinations, i.e., $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$ for all $\mathcal{S} \in \mathcal{P}(\mathcal{M})$. The number of unique feature combinations is 2^M, hence, the number of conditional distributions to model increases exponentially with the number of features. Thus, the methods above have to train 2^M different models, which eventually becomes computationally intractable. The method we propose in Section 3 comes from the machine learning community and can model all the 2^M different conditional distributions simultaneously with a single variational autoencoder.

3. Variational Autoencoder with Arbitrary Conditioning

The methodology of the *variational autoencoder with arbitrary conditioning* (`VAEAC`) (Ivanov et al., 2019) is an extension of the *conditional variational autoencoder* (`CVAE`) (Sohn et al., 2015), which is in turn a special type of a *variational autoencoder* (`VAE`) (Kingma and Welling, 2014; Rezende et al., 2014). If the concept of `VAE`s is unfamiliar to the reader, we recommend reading Kingma and Welling (2019, Ch. 1 & 2). These chapters give a motivational introduction to `VAE`s and lay a solid technical foundation which is necessary to understand the methodology of `VAEAC`.

The goal of the `VAE` is to give a probabilistic representation of the true unknown distribution $p(\boldsymbol{x})$. Briefly stated, the `VAE` assumes a *latent variable model* $p_{\boldsymbol{\xi}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\xi}}(\boldsymbol{z}) p_{\boldsymbol{\xi}}(\boldsymbol{x} | \boldsymbol{z})$, where $\boldsymbol{\xi}$ are the model parameters and $\boldsymbol{z}$ are the latent variables which conceptually represent an encoding of $\boldsymbol{x}$. Marginalizing over $\boldsymbol{z}$ gives $p_{\boldsymbol{\xi}}(\boldsymbol{x}) = \int p_{\boldsymbol{\xi}}(\boldsymbol{x}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z}$, which is an estimate of $p(\boldsymbol{x})$. The `CVAE` does the same for the conditional distribution $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$, for a specific subset $\mathcal{S} \in \mathcal{P}(\mathcal{M})$ of features. Here $\boldsymbol{x}_{\bar{\mathcal{S}}}$ is the complement to $\boldsymbol{x}_{\mathcal{S}}$, that is, the features of $\boldsymbol{x}$ not in $\boldsymbol{x}_{\mathcal{S}}$. The `VAEAC` generalizes this methodology to all conditional distributions $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$, for all possible feature subsets $\mathcal{S} \in \mathcal{P}(\mathcal{M})$ simultaneously.

In this paper, we present the `VAEAC` methodology for the Shapley value setting in the conventional Shapley value notation used in Section 2. The `VAEAC` consists of three fully connected feedforward neural networks (NNs): the *full encoder*[1] $p_{\boldsymbol{\varphi}}$, *masked encoder* $p_{\boldsymbol{\psi}}$,

[1] In *variational inference* notation, the variational distributions are often denoted by q instead of p.

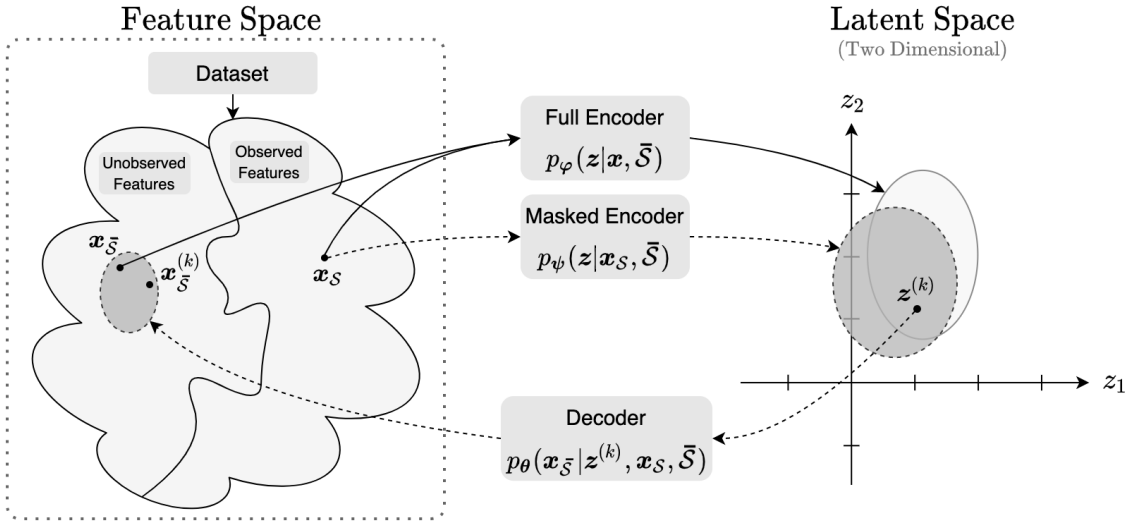

Figure 1: An abstract representation of how `VAEAC` generates the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ conditioned on the observed features $\boldsymbol{x}_{\mathcal{S}}$, for $k = 1, 2, \ldots, K$, where K denotes the number of Monte Carlo samples in (3). The feature space can, e.g., be the positive real numbers, as in Section 4.2. Conceptually, the full encoder guides the masked encoder in the *training phase* of the `VAEAC` method, while only the masked encoder and decoder are used in the *deployment phase* when generating the conditional samples.

and *decoder* $p_{\boldsymbol{\theta}}$.[2] The roles of the different entities are abstractly presented in Figure 1, which is an extension of Figure 2.1 in Kingma and Welling (2019), and they will be explained in greater detail in the following sections.

In Section 3.1, we give an overview of the `VAEAC` methodology. Sections 3.2 and 3.3 explain the *deployment phase* and *training phase* of the `VAEAC` method, respectively, where the former is when we generate the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$. Our novel non-uniform *masking scheme* is introduced in Section 3.4. Finally, `VAEAC`'s handling of *missing data* and use of *skip connections* is discussed in Sections 3.5 and 3.6, respectively.

3.1 Overview of the `VAEAC` Method

Let $p(\boldsymbol{x})$ denote the true, but unknown distribution of the data, where the input $\boldsymbol{x}$ is an M-dimensional vector of continuous and/or categorical features. The analytical expression for a general conditional distribution $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$ is unknown, except for some distributions $p(\boldsymbol{x})$, such as the multivariate Gaussian and Burr distributions. Here $\boldsymbol{x}_{\mathcal{S}}$ and $\boldsymbol{x}_{\bar{\mathcal{S}}}$ denote the *observed features* (that we condition on) and the *unobserved features* (that we need to infer) of the *full* sample $\boldsymbol{x}$, respectively. Ivanov et al. (2019) say that the unobserved features $\boldsymbol{x}_{\bar{\mathcal{S}}}$ are *masked* by the *mask* $\bar{\mathcal{S}}$. We will use mask $\bar{\mathcal{S}}$ and coalition $\mathcal{S}$ interchangeably as they are uniquely related by $\bar{\mathcal{S}} = \mathcal{M} \backslash \mathcal{S}$.

The `VAEAC` model gives a probabilistic representation $p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ of the true unknown conditional distribution $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$, for a given $\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*$ and an arbitrary coalition

[2] Ivanov et al. (2019) call them the *proposal network*, *prior network*, and *generative network*, respectively.

$\mathcal{S} \in \mathcal{P}(\mathcal{M})$. Here, $\boldsymbol{\psi}$ and $\boldsymbol{\theta}$ denote the learned model parameters of the masked encoder $p_{\boldsymbol{\psi}}$ and decoder $p_{\boldsymbol{\theta}}$, respectively. The masked encoder is an extension of the `VAE` methodology, which solely consists of a decoder and a full encoder $p_{\boldsymbol{\varphi}}$, where the latter is parameterized by $\boldsymbol{\varphi}$. Both encoders are stochastic mappings from parts of the feature space to a latent space. The full encoder $p_{\boldsymbol{\varphi}}$ stochastically maps the *full* observation $\boldsymbol{x} = \{\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}}\}$ to a latent representation $\boldsymbol{z}$, hence, the name full encoder. The masked encoder $p_{\boldsymbol{\psi}}$ does the same but only conditioned on the observed features $\boldsymbol{x}_{\mathcal{S}}$. The unobserved features $\boldsymbol{x}_{\bar{\mathcal{S}}}$ of $\boldsymbol{x}$ have been *masked*, hence, the name masked encoder. The decoder stochastically maps latent representations $\boldsymbol{z}^{(k)}$ back to the unobserved part of the feature space, from which we can generate on-distribution samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$. These samples are then used to estimate the contribution function $v(\mathcal{S})$ in (3).

The full encoder is only used during the *training phase* of the `VAEAC` method when we have access to the full training observation $\boldsymbol{x}$. Conceptually, the purpose of the full encoder is to guide the masked encoder in finding proper latent representations. In the training phase, we artificially mask out coalitions of features from $\boldsymbol{x}$ and train `VAEAC` to generate likely values of the artificially masked/unobserved features $\boldsymbol{x}_{\bar{\mathcal{S}}}$ conditioned on the observed features $\boldsymbol{x}_{\mathcal{S}}$. In the *deployment phase*, the `VAEAC` method only uses the masked encoder and decoder to generate the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ used in (3). This can be seen by following the arrows in Figure 2, where we give a detailed visual representation of a `VAEAC` model and its associated NNs, which should make the explanations of the `VAEAC` approach easier to follow. We will go through an example based on Figure 2 once all notation and terminology is introduced.

Since `VAEAC` estimates all conditional distributions simultaneously, the trained conditional model $p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ may be more precise for some $\bar{\mathcal{S}}$ and less precise for others. The precision can be controlled by introducing a *masking scheme*, which is a distribution $p(\bar{\mathcal{S}})$ over different masks/feature subsets $\bar{\mathcal{S}}$. The mask distribution $p(\bar{\mathcal{S}})$ can be arbitrary, or defined based on the problem at the hand. However, full support over $\mathcal{P}(\mathcal{M})$ is crucial to ensure that $p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ can evaluate arbitrary conditioning. Masking schemes and possible applications are further discussed in Sections 3.4 and 6.

The log-likelihood objective function of the `VAEAC` model is

$$\mathbb{E}_{p(\boldsymbol{x})}\mathbb{E}_{p(\bar{\mathcal{S}})}\big[\log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})\big], \tag{4}$$

which is to be maximized by tuning $\boldsymbol{\psi}$ and $\boldsymbol{\theta}$. When all features are unobserved and need to be inferred, i.e., $\mathcal{S} = \emptyset \Leftrightarrow \bar{\mathcal{S}} = \mathcal{M}$ as we do not condition on any feature, `VAEAC` estimates the logarithm of $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = p(\boldsymbol{x}_{\mathcal{M}}|\boldsymbol{x}_{\emptyset}, \mathcal{M}) = p(\boldsymbol{x})$. That is, `VAEAC` models the full joint data distribution, which corresponds to the objective of the original `VAE` (Kingma and Welling, 2014; Rezende et al., 2014). Furthermore, if $p(\bar{\mathcal{S}}) = 1$ for some fixed $\bar{\mathcal{S}}$ (that is, one specific mask/distribution) then (4) corresponds to the objective function of `CVAE` (Sohn et al., 2015). Thus, `VAEAC` generalizes both the `VAE` and `CVAE`.

3.2 The Deployment Phase of the `VAEAC` Method

The generative process of the `VAEAC` method is a two-step procedure, similar to that of `VAE` and `CVAE`, and constitutes the *deployment phase* where we generate the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}$ in (3). First, we generate a latent representation $\boldsymbol{z} \sim p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) \in \mathbb{R}^d$ using

the masked encoder, where the dimension of the latent space, d, is specified by the user of `VAEAC`. Second, we sample the unobserved features from $\boldsymbol{x}_{\bar{\mathcal{S}}} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ using the decoder, as illustrated in Figure 1 and Figure 2. This process marginalizes over the latent variables, resulting in the following distribution over the unobserved features:

$$p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})}\big[p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})\big], \tag{5}$$

which is similar to the procedures in `VAE`s and `CVAE`s.

The continuous latent variables $\boldsymbol{z}$ of the masked encoder are assumed to be distributed according to a multivariate Gaussian distribution with a diagonal covariance matrix, that is, $p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \mathcal{N}_d\big(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})]\big)$. Here $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$ are the d-dimensional outputs of the NN for the masked encoder $p_{\boldsymbol{\psi}}$. As the output nodes of NNs are unbounded, we apply the soft-plus function[3] to $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$ to ensure that the standard deviations are positive. The features go through an *internal* encoding layer, before entering the masked encoder, where the continuous features of $\boldsymbol{x}_{\mathcal{S}}$ are directly passed through, while the categorical features are one-hot encoded, see Appendix C.1. One-hot encoding is the most widely used coding scheme for categorical features in NNs (Potdar et al., 2017). Note that the components of the latent representation $\boldsymbol{z}$ are conditionally independent given $\boldsymbol{x}_{\mathcal{S}}$ and $\bar{\mathcal{S}}$.

The full encoder is also assumed to be a multivariate Gaussian distribution with a diagonal covariance matrix, that is, $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}, \bar{\mathcal{S}}) = \mathcal{N}_d\big(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}, \bar{\mathcal{S}}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}, \bar{\mathcal{S}})]\big)$. The full encoder is conditioned on the full observation $\boldsymbol{x}$, in contrast to the masked encoder which is solely conditioned on the observed features $\boldsymbol{x}_{\mathcal{S}}$. Furthermore, the features go through the internal encoding layer before entering the full encoder. The full encoder is only used during the training phase of the `VAEAC` method, which will become clear in Section 3.3.

The decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ is also assumed to follow a multivariate Gaussian structure with a diagonal covariance matrix for the continuous features of $\boldsymbol{x}_{\bar{\mathcal{S}}}$, that is, $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \mathcal{N}\big(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})]\big)$. We discuss this assumption in Section 6. As the latent variables $\boldsymbol{z}$ are continuous and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ is in this case Gaussian, the distribution $p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ in (5) can be seen as an infinite mixture of Gaussian distributions (Kingma and Welling, 2019, p. 12), which is a universal approximator of densities, in the sense that any smooth density can be arbitrarily well approximated (Goodfellow et al., 2016, p. 65). We illustrate this property in Appendix G.

The Gaussian structure is incompatible with categorical features. Assume for now that x_j is categorical with L categories. When working with multi-class features and NNs, it is common practice to let the outputs $\boldsymbol{w}_{\boldsymbol{\theta},j}(\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ of the NN be the logits of the probabilities for each category. That is, $\boldsymbol{w}_{\boldsymbol{\theta},j}(\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \text{logit}(\boldsymbol{p}_j) = \log[\boldsymbol{p}_j/(1-\boldsymbol{p}_j)]$ is an L-dimensional vector of the logits for each of the L categories for the categorical jth feature. As logits are unbounded, we use the softmax function[4] to map the logits to probabilities in $(0, 1)$. Finally, we use the categorical distribution with L categories to model each of the categorical features, that is, $x_j \sim \text{Cat}\big(\text{Softmax}(\boldsymbol{w}_{\boldsymbol{\theta},j}(\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}))\big)$. Note that even though the categorical features are internally one-hot encoded, the sampled categorical features from the decoder will be in the original categorical form and *not* in the one-hot encoded representation. Furthermore, note that the components of $\boldsymbol{x}_{\bar{\mathcal{S}}}$ are conditionally independent given $\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}$, and $\bar{\mathcal{S}}$.

[3] $\text{softplus}(x) = \log(1 + \exp(x))$.

[4] $\text{softmax}(\boldsymbol{w}_{\boldsymbol{\theta},j})_i = \exp\{\boldsymbol{w}_{\boldsymbol{\theta},j,i}\}/\sum_{l=1}^{L}\exp\{\boldsymbol{w}_{\boldsymbol{\theta},j,l}\}$, for $i = 1, \ldots, L$.

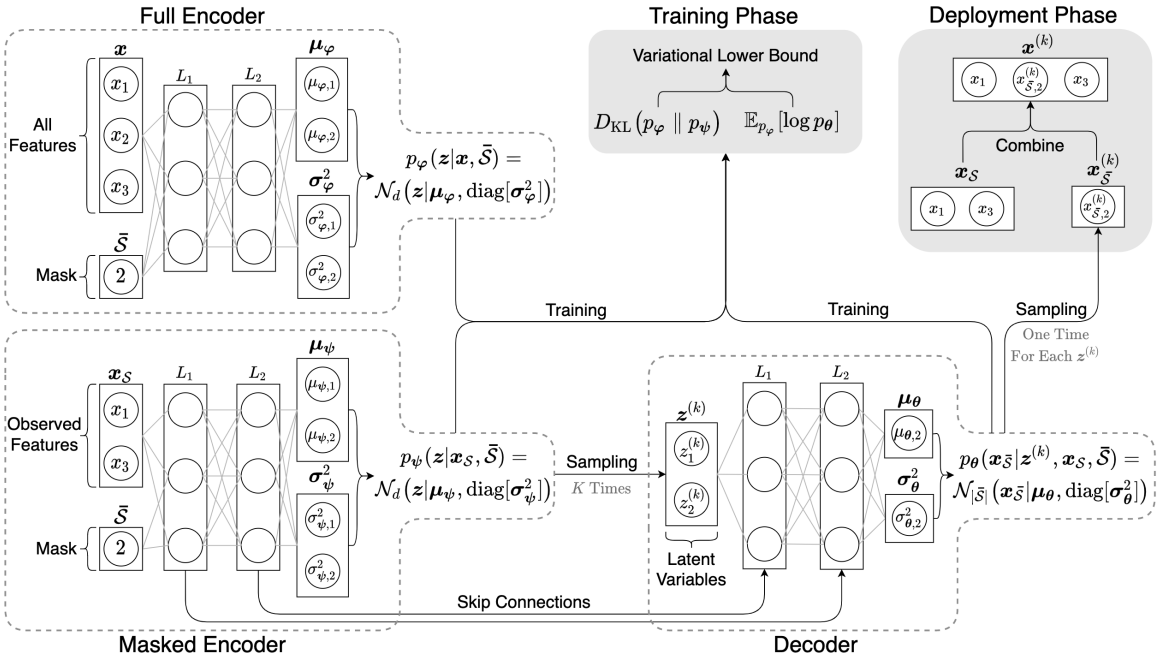

Figure 2: A visual representation of the `VAEAC` model used in the example given at the end of Section 3.2. The model has parameters `depth` $= 2$, `width` $= 3$, $d =$ `latent_dim` $= 2$ for an $M = 3$-dimensional continuous input $\boldsymbol{x} = \{x_1, x_2, x_3\}$ with coalition $\mathcal{S} = \{1, 3\}$ and mask $\bar{\mathcal{S}} = \{2\}$. All networks are used during the training phase, but only the masked encoder and decoder are used in the deployment phase to generate the conditional samples $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$, for $k = 1, ..., K$, where K is the number of Monte Carlo samples in (3).

We now give a walk-through of the `VAEAC` model illustrated in Figure 2 for $M = 3$-dimensional continuous data. The notation and representation of the network architecture is simplified for ease of understanding. The complete implementation details are found in Appendix C.1. The aim is to sample $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)} \sim p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$, for $k = 1, ..., K$, where K is the number of Monte Carlo samples in (3). For the $M = 3$-dimensional setting, there are $2^M = 8$ possible coalitions $\mathcal{S}$, that is, $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$. However, the edge cases are treated differently in the R-package `shapr` (Sellereite and Jullum, 2019), which we use to compute the Shapley values, as no modeling is needed. For $\mathcal{S} = \mathcal{M} = \{1, 2, 3\}$, we have that $\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*$ and $v(\mathcal{M}) = f(\boldsymbol{x}^*)$ by definition. For the other edge case, $\mathcal{S} = \emptyset$, we have by definition that $\phi_0 = v(\emptyset) = \mathbb{E}[f(\boldsymbol{x})]$, which `shapr` lets the user specify. A common estimate is the average training response (Aas et al., 2021a), which we also use in this article. In Figure 2, we illustrate the `VAEAC` method when given coalition $\mathcal{S} = \{1, 3\}$, that is, mask $\bar{\mathcal{S}} = \{2\}$, but the procedure is identical for the other coalitions.

Let $\boldsymbol{x} = \{x_1, x_2, x_3\}$ denote the full observation, where the associated observed and unobserved feature vectors are $\boldsymbol{x}_{\mathcal{S}} = \{x_1, x_3\}$ and $\boldsymbol{x}_{\bar{\mathcal{S}}} = \{x_2\}$, respectively. We start by sending $\boldsymbol{x}_{\mathcal{S}}$ and $\bar{\mathcal{S}}$ through the masked encoder, which yields the latent probabilistic representation

$p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}) = \mathcal{N}_d\big(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}),\operatorname{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})]\big)$, where $d = 2$. We sample K latent representations $\boldsymbol{z}^{(k)} \sim p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})$, which are then sent through the decoder. This yields K different distrbutions $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}) = \mathcal{N}\big(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}),\operatorname{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})]\big)$, as the means $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})$ and variances $\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})$ are functions of $\boldsymbol{z}^{(k)}$. When $K \to \infty$, we obtain an infinite mixture of Gaussian distributions, see Appendix G. We sample *one* observation from each inferred distribution, that is, $\boldsymbol{x}^{(k)}_{\bar{\mathcal{S}}} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}^{(k)},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})$. Sampling more values is also possible. Ivanov et al. (2019) call this process of repeatedly generating the missing values $\boldsymbol{x}_{\bar{\mathcal{S}}}$ based on $\boldsymbol{x}_{\mathcal{S}}$ for *missing features multiple imputation*.

3.3 The Training Phase of the VAEAC Method

The model parameters $\boldsymbol{\psi}$ and $\boldsymbol{\theta}$ are not easily tuned by maximizing the log-likelihood (4), as this optimization problem is challenging due to intractable posterior inference. Kingma and Welling (2019, Ch. 2) discuss the same issue for the VAE. The optimization problem is bypassed by *variational inference*[5] (Blei et al., 2017) where we rather optimize the *variational lower bound* (VLB). The VLB is also called the *evidence lower bound* (ELBO) in variational inference. The main contribution of Kingma and Welling (2014) is their reparameterization trick of VAE's latent variables. This allows for efficient fitting of VAE's model parameters and optimization of the VLB by using backpropagation and stochastic gradient descent (Goodfellow et al., 2016, Ch. 6 & 8). The same type of techniques are used to optimize VAEAC's model parameters.

Like for VAEs, we can derive a variational lower bound for VAEAC:

$$\begin{aligned}
\log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}) &= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] + \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] \\
&\geq \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\right] - D_{\mathrm{KL}}\big(p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}}) \parallel p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\big) \\
&= \mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x},\bar{\mathcal{S}}|\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi}),
\end{aligned} \tag{6}$$

where the Kullback–Leibler divergence, D_{KL}, for two continuous probability distributions p and q, is given by $D_{\mathrm{KL}}\big(p \parallel q\big) = \int_{-\infty}^{\infty} p(\boldsymbol{z}) \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})}\,\mathrm{d}\boldsymbol{z} = \mathbb{E}_{p(\boldsymbol{z})}\left[\log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})}\right] \geq 0$. We obtain a *tight* variational lower bound if $p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})$ is close to $p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})$ with respect to the Kullback–Leibler divergence. The full derivation of (6) is presented in Appendix B.

Instead of maximizing the log-likelihood in (4), we rather have the following variational lower bound optimization problem for VAEAC:

$$\max_{\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi}} \mathbb{E}_{p(\boldsymbol{x})}\mathbb{E}_{p(\bar{\mathcal{S}})}\left[\mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x},\bar{\mathcal{S}}|\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi})\right] = \min_{\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi}} \mathbb{E}_{p(\boldsymbol{x})}\mathbb{E}_{p(\bar{\mathcal{S}})}\left[-\mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x},\bar{\mathcal{S}}|\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi})\right], \tag{7}$$

which is optimized similarly as VAEs. Since we assumed $p_{\boldsymbol{\varphi}}$ and $p_{\boldsymbol{\psi}}$ to be multivariate Gaussians, we have a closed-form expression for the Kullback–Leibler divergence in (6). The parameters $\boldsymbol{\psi}$ and $\boldsymbol{\theta}$ can be directly optimized using Monte Carlo estimates, backpropagation, and stochastic gradient descent.

The gradient of $\mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x},\bar{\mathcal{S}}|\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi})$ with respect to the variational parameter $\boldsymbol{\varphi}$ is more complicated to compute, as the expectation in (6) is taken with respect to $p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})$, which

[5] Also called *variational Bayes* and VAEs belong to the family of *variational Bayesian methods*.

is a function of $\boldsymbol{\varphi}$. However, the gradient can be efficiently estimated using the reparameterization trick of Kingma and Welling (2014). The trick is to rewrite the latent variable vector as $\boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}, \bar{\mathcal{S}}) + \boldsymbol{\epsilon}\boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x}, \bar{\mathcal{S}})$, where $\boldsymbol{\epsilon} \sim \mathcal{N}_d(0, I)$ and $\boldsymbol{\mu}_{\boldsymbol{\varphi}}$ and $\boldsymbol{\sigma}_{\boldsymbol{\varphi}}$ are deterministic functions parameterized by the NN of the full encoder. That is, the randomness in $\boldsymbol{z}$ has been *externalized* to $\boldsymbol{\epsilon}$ by reparameterizing $\boldsymbol{z}$ as a deterministic and differentiable function of $\boldsymbol{\varphi}$, $\boldsymbol{x}$, $\bar{\mathcal{S}}$, and the random variable $\boldsymbol{\epsilon}$. Thus, the gradient is

$$\begin{aligned}\frac{\partial \mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x}, \bar{\mathcal{S}}|\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\varphi})}{\partial \boldsymbol{\varphi}} = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}_d(0,I)} \left[\frac{\partial}{\partial \boldsymbol{\varphi}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{\mu}_{\boldsymbol{\varphi}}(\boldsymbol{x}, \bar{\mathcal{S}}) + \boldsymbol{\epsilon}\boldsymbol{\sigma}_{\boldsymbol{\varphi}}(\boldsymbol{x}, \bar{\mathcal{S}}), \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) \right] \\ - \frac{\partial}{\partial \boldsymbol{\varphi}} D_{\mathrm{KL}}\big(p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}, \bar{\mathcal{S}}) \parallel p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})\big),\end{aligned}$$

where the first term is estimated using Monte Carlo sampling, while we use the closed-form expression mentioned above for the second term.

3.3.1 Prior in Latent Space

When optimizing the variational lower bound in (7), there are no restrictions on the size of the estimated mean $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and standard deviation $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$ in the masked encoder. That is, they can grow to infinity and cause numerical instabilities. We follow Ivanov et al. (2019) and solve this problem by adding a normal prior on $\boldsymbol{\mu}_{\boldsymbol{\psi}}$ and gamma prior on $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$ to prevent divergence. That is, we introduce a restricted version of $p_{\boldsymbol{\psi}}(\boldsymbol{z}, \boldsymbol{\mu}_{\boldsymbol{\psi}}, \boldsymbol{\sigma}_{\boldsymbol{\psi}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$:

$$p^{\mathrm{res}}_{\boldsymbol{\psi}}(\boldsymbol{z}, \boldsymbol{\mu}_{\boldsymbol{\psi}}, \boldsymbol{\sigma}_{\boldsymbol{\psi}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \mathcal{N}_d\big(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}), \mathrm{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\psi}}(\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})]\big)\mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{\psi}}|0, \sigma^2_{\mu}\big)\Gamma(\boldsymbol{\sigma}_{\boldsymbol{\psi}}|1 + \sigma_{\sigma}^{-1}, \sigma_{\sigma}^{-1}).$$

Here Γ is the gamma distribution with shape parameter $1 + \sigma_{\sigma}^{-1}$ and rate parameter σ_{σ}^{-1}, that is, the mean is $1 + \sigma_{\sigma}$ and the variance is $\sigma_{\sigma}(1 + \sigma_{\sigma})$. The hyperparameters σ_{μ} and σ_{σ} are set to be large, so as not to affect the learning process significantly.

When rederiving the variational lower bound (6) with $p^{\mathrm{res}}_{\boldsymbol{\psi}}$ instead of $p_{\boldsymbol{\psi}}$, we get two additional regularizing terms in the VLB, after removing fixed constants. Thus, we maximize the following VLB in (7):

$$\begin{aligned}\mathcal{L}^{\mathrm{res}}_{\texttt{VAEAC}}(\boldsymbol{x}, \bar{\mathcal{S}}|\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\varphi}) = \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}, \bar{\mathcal{S}})} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})\right] - D_{\mathrm{KL}}\big(p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}, \bar{\mathcal{S}}) \parallel p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})\big) \\ - \frac{\boldsymbol{\mu}^2_{\boldsymbol{\psi}}}{2\sigma^2_{\mu}} + \frac{1}{\sigma_{\sigma}}(\log(\boldsymbol{\sigma}_{\boldsymbol{\psi}}) - \boldsymbol{\sigma}_{\boldsymbol{\psi}}).\end{aligned}$$

Large values of $\boldsymbol{\mu}^2_{\boldsymbol{\psi}}$ and $\boldsymbol{\sigma}_{\boldsymbol{\psi}}$ decrease the new VLB through the last two terms, but their magnitude is controlled by the hyperparameters σ_{μ} and σ_{σ}. Ivanov et al. (2019, Eq. 8) obtain identical regularizers, but there is a misprint in their shape parameter of the gamma prior.

3.4 Masking Schemes

The default version of the `VAEAC` method uses the uniform masking scheme, i.e., $p(\bar{\mathcal{S}}) = 2^{-M}$. This means that we can use a Bernoulli(0.5) to model whether feature j is in $\mathcal{S}$ or not. As $p(\bar{\mathcal{S}})$ has full support over $\mathcal{P}(\mathcal{M})$, the `VAEAC` method can evaluate arbitrary conditioning. However, we can make `VAEAC` focus more of its estimation efforts on certain coalitions by

defining a non-uniform masking scheme. Recall that the mask $\bar{\mathcal{S}}$ and the corresponding coalition $\mathcal{S}$ are uniquely related by $\bar{\mathcal{S}} = \mathcal{M} \backslash \mathcal{S}$.

In high-dimensional settings, it is common to compute Shapley values based on a set $\mathcal{C}$ of coalitions sampled from $\mathcal{P}(\mathcal{M})$ with replacements (Lundberg and Lee, 2017; Williamson and Feng, 2020; Aas et al., 2021a; Frye et al., 2021; Mitchell et al., 2022). In these settings, we can specify a masking scheme based on the sampling frequency of the coalitions in $\mathcal{C}$. More precisely, we let $p(\bar{\mathcal{S}}) = p(\mathcal{M} \backslash \mathcal{S}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathbf{1}(c = \mathcal{S})$, where $\mathbf{1}$ is the indicator function, and we denote the corresponding `VAEAC` method by $\texttt{VAEAC}_{\mathcal{C}}$. Thus, the $\texttt{VAEAC}_{\mathcal{C}}$ approach will, in contrast to the regular `VAEAC` approach, *not* try to learn the conditional distributions $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}})$ for all $\mathcal{S} \in \mathcal{P}(\mathcal{M})$, but rather only for the coalitions in $\mathcal{C}$. If coalition $\mathcal{S}$ has been sampled twice as often as $\mathcal{S}'$, then the training samples will be twice as often artificially masked by $\bar{\mathcal{S}}$ than $\bar{\mathcal{S}}'$ during the training phase of the $\texttt{VAEAC}_{\mathcal{C}}$ method. That is, the $\texttt{VAEAC}_{\mathcal{C}}$ approach trains twice as much on learning $p(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{x}_{\mathcal{S}})$ compared to $p(\boldsymbol{x}_{\bar{\mathcal{S}}'} | \boldsymbol{x}_{\mathcal{S}'})$. The hyperparameters of $\texttt{VAEAC}_{\mathcal{C}}$ are the same as those described for `VAEAC` in Appendix C.2. In Section 4, we will see that this new non-uniform masking scheme significantly improves the `VAEAC` methodology in the high-dimensional simulation studies.

3.5 Missing Feature Values

Incomplete data is common for real-world data sets, that is, some feature values for some observations are missing, denoted by `NA`. Many classical statistical models rely on complete data in their training phase, e.g., linear models and GAMs, while some modern ML models handles missing data, e.g., XGBoost (Chen et al., 2015). Consider a model of the latter type trained on incomplete data. In the Shapley value explanation framework, the approach used to estimate the feature dependencies should also handle incomplete data in the training phase to fully utilize all available data. In contrast to the competing conditional approaches used in Section 4, the `VAEAC` approach directly supports incomplete training data. The original purpose of the `VAEAC` method was to generate multiple imputations for missing entries in incomplete data sets, while simultaneously preserving the feature dependencies.

We do not investigate `VAEAC`'s handling of missing data in the current article, as we want a fair comparison with the competing methods, which lack direct support for missing data. However, we give a brief outline of how `VAEAC` handles missing feature values in the training phase below. Additionally, in Section 6, we discuss how the `VAEAC` method can be extended to also support incomplete data in the deployment phase.

We refer to Ivanov et al. (2019) for the fine details and evaluation of `VAEAC`'s performance when applied on incomplete training data. However, the main points are the following. First, `VAEAC` ensures that the missing feature values are always unobserved in the training phase by masking them on an individual basis. Hence, the mask distribution must be conditioned on $\boldsymbol{x}$, that is, $p(\bar{\mathcal{S}})$ turns into $p(\bar{\mathcal{S}} | \boldsymbol{x})$ in (4) and (7). Second, the `NA` values of $\boldsymbol{x}_{\bar{\mathcal{S}}}$ must be excluded when computing the $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ term in (6). This is solved by marginalizing out the missing features, that is, $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}} | \boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) = \sum_{i: i \in \bar{\mathcal{S}}, x_i \neq \texttt{NA}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_i | \boldsymbol{z}, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$. Finally, to ensure that the full encoder can distinguish between missing and masked features during training, Ivanov et al. (2019) add an additional *missing feature mask* to the full encoder. That is, the full encoder receives the missing feature mask, the unobserved/masked feature mask $\bar{\mathcal{S}}$, and the training instance $\boldsymbol{x}$ where `NA` values are set to zero.

3.6 Skip Connections

Our `VAEAC` approach to Shapley value estimation includes skip connections with concatenation between the layers of the masked encoder and the decoder, as seen in Figure 2. This architectural choice is influenced by Ivanov et al. (2019), who report that they saw an improvement in the log-likelihood when they included skip connections in the `VAEAC` model. Skip connections are a standard tool in deep learning for improving the performance of neural networks and are extensively used in many of the well-cited network architectures, e.g., ResNet (He et al., 2016), U-Net (Ronneberger et al., 2015), and DenseNet (Huang et al., 2017). The motivation is that the skip connections yield direct gradient flows between non-succeeding layers, which counteracts the *vanishing gradient problem* of deep neural networks in the backpropagation step. Therefore, skip connections are one of several possible architectural tools (other tools are activation functions such as LeakyReLU) used to ensure effective learning of all model parameters, and they can stabilize the training and convergence (Mao et al., 2016; Li et al., 2018).

4. Simulation Studies

A major problem of evaluating explanation methods is that there is in general no ground truth for authentic real-world data. In this section, we simulate data for which we can compute the true Shapley values (1). We then compare how close the estimated Shapley values are to the corresponding true ones. However, this evaluation methodology is intractable in high-dimensional settings due to the exponential number of terms in the Shapley value formula. Furthermore, the expectations in (2) rely on solving integrals of increasing dimensions which requires more Monte Carlo samples in (3) to maintain the precision. In high-dimensional settings, we rather compare how close a sampled set of estimated contribution functions (3) are to the true counterparts (2).

We conduct and discuss two types of simulation studies: one for continuous data (Section 4.2) and one with mixed data (Section 4.3). For the sake of completeness, a simulation study limited to categorical features has also been investigated, but as real-world data sets are seldom restricted to only categorical features, we rather report the corresponding results in Appendix D. In both simulation studies, we first look at exact low-dimensional settings before progressing to sampled high-dimensional settings.

For the continuous data setting, we compare our `VAEAC` approach to Shapley value estimation with the `independence` method from Lundberg and Lee (2017), the `empirical`, `Gaussian`, and `copula` approaches from Aas et al. (2021a), and the `ctree` method from Redelmeier et al. (2020). These methods are briefly described in Appendix A. For the mixed data setting with continuous and categorical features, we dismiss the methods of Aas et al. (2021a) as they lack support for categorical data. We can make their methods work by using feature encodings, such as one-hot encoding, before applying the methods. We call this *external* feature encoding. However, we deem external feature encoding infeasible, as it drastically increases the computational time (Redelmeier et al., 2020, q.v. Tables 4 and 6). This is due to the need to estimate Shapley values for each dummy feature before combining them to form the Shapley values of the original categorical features. Note that `VAEAC`'s use of *internal* encoding layers, discussed in Section 3.2, does not affect `VAEAC`'s capability to receive and generate categorical data and thereby direct computation of the

Shapley values for categorical features. In the high-dimensional simulation studies, we also include the novel $\texttt{VAEAC}_{\mathcal{C}}$ approach described in Section 3.4.

All mentioned methodologies are implemented in the software package `shapr` (Sellereite and Jullum, 2019, version 0.2.0) in the R programming language (R Core Team, 2020), with default hyperparameters. See Appendix C for the choice of `VAEAC`'s hyperparameters. Ivanov et al. (2019) have implemented `VAEAC` in Python (Python Core Team, 2020), see `https://github.com/tigvarts/vaeac`. We have made changes to their implementation and used `reticulate` (Ushey et al., 2020) to run Python code in R, such that our `VAEAC` and $\texttt{VAEAC}_{\mathcal{C}}$ approaches to Shapley value estimation run on top of the `shapr`-package. Our version of `VAEAC` and $\texttt{VAEAC}_{\mathcal{C}}$ can be accessed on `https://github.com/LHBO/ShapleyValuesVAEAC`.

4.1 Evaluation Criteria

We consider three types of evaluation criteria which we describe in this section. The first, EC1, is the mean absolute error[6] (MAE), averaged over all test observations and features:

$$\text{EC1} = \text{MAE}_{\phi}(\text{method } \texttt{q}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{1}{M} \sum_{j=1}^{M} |\phi_{j,\texttt{true}}(\boldsymbol{x}^{[i]}) - \hat{\phi}_{j,\texttt{q}}(\boldsymbol{x}^{[i]})|. \tag{8}$$

Here, $\boldsymbol{\phi}_{\texttt{true}}$ and $\hat{\boldsymbol{\phi}}_{\texttt{q}}$ are the true Shapley values and the Shapley values computed using method `q`, respectively. The true Shapley values are not known in general. However, in our simulation studies, the conditional distribution function $p_{\texttt{true}}$ is analytically known and samplable. This means that we can compute the true contribution function $v_{\texttt{true}}(\mathcal{S})$ in (2) by sampling $\boldsymbol{x}^{(k)}_{\bar{\mathcal{S}},\texttt{true}} \sim p_{\texttt{true}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}})$ and using (3), for $\mathcal{S} \in \mathcal{P}(\mathcal{M})$. The true Shapley values are then obtained by inserting the $v_{\texttt{true}}(\mathcal{S})$ quantities into the Shapley formula (1). The $v_{\texttt{true}}(\mathcal{S})$ can be arbitrarily precise by choosing a sufficiently large number of Monte Carlo samples K. The Shapley values for method `q` are computed in the same way, but using the conditional distributions estimated using approach `q` instead of the true ones.

The second criterion, EC2, measures the mean squared error between $v_{\texttt{true}}$ and $v_{\texttt{q}}$, i.e.,

$$\text{EC2} = \text{MSE}_{v}(\text{method } \texttt{q}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{1}{N_{\mathcal{S}}} \sum_{\mathcal{S} \in \mathcal{P}^*(\mathcal{M})} \Big(v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}^{[i]}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}^{[i]}) \Big)^2. \tag{9}$$

Here $\mathcal{P}^*(\mathcal{M}) = \mathcal{P}(\mathcal{M}) \backslash \{\emptyset, \mathcal{M}\}$, that is, we skip $v(\emptyset)$ and $v(\mathcal{M})$ as they are the same regardless of which method `q` that is used. Thus, $N_{\mathcal{S}} = |\mathcal{P}^*(\mathcal{M})| = 2^M - 2$.

The final criterion, EC3, is given by

$$\text{EC3} = \text{EPE}_{v}(\text{method } \texttt{q}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{1}{N_{\mathcal{S}}} \sum_{\mathcal{S} \in \mathcal{P}^*(\mathcal{M})} \Big(f(\boldsymbol{x}^{[i]}) - \hat{\boldsymbol{v}}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}^{[i]}) \Big)^2, \tag{10}$$

which is based on (9) being an estimator of

$$\mathbb{E}_{\mathcal{S}}\mathbb{E}_{\boldsymbol{x}}(v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}))^2 = \mathbb{E}_{\mathcal{S}}\mathbb{E}_{\boldsymbol{x}}(f(\boldsymbol{x}) - \hat{v}_{\texttt{q}}(\mathcal{S}, \boldsymbol{x}))^2 - \mathbb{E}_{\mathcal{S}}\mathbb{E}_{\boldsymbol{x}}(f(\boldsymbol{x}) - v_{\texttt{true}}(\mathcal{S}, \boldsymbol{x}))^2, \tag{11}$$

[6] Using the mean squared error yields the same average ranking of the methods in the simulation studies.

where the decomposition follows from Covert et al. (2020, Appendix A). The first term on the right hand side of (11) corresponds to (10), while the second term is a fixed (unknown) constant not influenced by approach `q`.

An advantage of the EC3 is that $v_{\texttt{true}}$ is not involved, which means that it also may be used for real-world data. However, this criterion has two main drawbacks. First, it can only be used to rank different approaches and not assess how close they are to the optimal method. This is because we do not know the minimum value of this criterion. The second drawback, which it has in common with the EC2, is that it evaluates contribution functions and not Shapley values. The estimates for $v(\mathcal{S})$ can overshoot for some coalitions and undershoot for others, and such errors may cancel each other out in the Shapley formula.

In high-dimensional settings, it is intractable to consider all 2^M coalitions $\mathcal{S} \in \mathcal{P}(\mathcal{M})$ in (8)–(10). In such cases, we sample $N_{\mathcal{S}}$ coalitions $\mathcal{S}$ from $\mathcal{P}(\mathcal{M})$ with replacement and weights equal to $\frac{|\mathcal{S}|!(M-|\mathcal{S}|-1)!}{M!}$, that is, the weights in the Shapley value formula (1). We use this subset of coalitions, denoted by $\mathcal{C}$, instead of $\mathcal{P}(\mathcal{M})$ when computing the three evaluation criteria.

4.2 Simulation Study: Continuous Data

For the continuous simulation study, we generate dependent features from the multivariate Burr distribution, which is briefly explained in Appendix E.1. The Burr distribution is skewed, strictly positive, and has known conditional distributions. Thus, we can compute the true Shapley values with arbitrary precision by using Monte Carlo integration. This simulation study is an extension of the one conducted in Aas et al. (2021b).

The full description of the data generating process is given in Appendix E.1, but the essentials are presented here. We sample N_{train} training and N_{test} test observations from a Burr$(\kappa, \boldsymbol{b}, \boldsymbol{r})$ distribution. The scale parameter κ is set to 2, which yields an average pair-wise Kendall's τ coefficient of 0.20, that is, moderate feature dependence. We sample $\boldsymbol{b}, \boldsymbol{r} \in \mathbb{R}^M$ from plausible values to make the setup work for any M divisible by five. We generate the response y according to an extension of the non-linear and heteroscedastic function used in Aas et al. (2021b), which was inspired by Chang and Joe (2019). That is,

$$y = \sum_{k=0}^{(M-5)/5} [\sin(\pi c_{3k+1} u_{5k+1} u_{5k+2}) + c_{3k+2} u_{5k+3} \exp\{c_{3k+3} u_{5k+4} u_{5k+5}\}] + \varepsilon, \tag{12}$$

where ε is added noise, c_l is a sampled coefficient for $l = 1, \ldots, \frac{3M}{5}$, and $u_j = F_j(x_j)$, for $j = 1, 2, \ldots, M$. Here $\boldsymbol{x}$ is multivariate Burr distributed and F_j is the true parametric (cumulative) distribution function for the jth feature x_j. Thus, each u_j is uniformly distributed between 0 and 1, but the dependence structure between the features is kept.

We let the predictive function f be a random forest model with 500 trees fitted using the R-package `ranger` (Wright and Ziegler, 2017) with default parameters. We use the different approaches to compute the three evaluation criteria. We repeat this setup R times and report the average EC values, i.e., the quality of the approaches is evaluated based on a total of $R \times N_{\text{test}}$ test observations. For each approach, we use $K = 250$ Monte Carlo samples to estimate the contribution functions in (3). We experimented with larger values of K and saw a marginal increase in accuracy, but at the cost of increased computation time,

M	$N_{\mathcal{S}}$	N_{train}	Crit.	Methods						
				`True`	`Indep.`	`Empir.`	`Gauss.`	`Copula`	`Ctree`	`VAEAC`
5	2^5	100	EC1	—	0.0967	0.1051	0.0534	0.0568	0.0829	**0.0492**
			EC2	—	0.1082	0.1242	0.0248	0.0274	0.0684	**0.0184**
			EC3	0.3041	0.4107	0.4333	0.3305	0.3310	0.3624	**0.3199**
		1000	EC1	—	0.1095	0.0479	0.0424	0.0472	0.0480	**0.0264**
			EC2	—	0.1579	0.0499	0.0159	0.0257	0.0325	**0.0092**
			EC3	0.4285	0.5821	0.4864	0.4490	0.4575	0.4615	**0.4402**
		5000	EC1	—	0.1157	0.0279	0.0426	0.0473	0.0338	**0.0239**
			EC2	—	0.1915	0.0183	0.0171	0.0245	0.0182	**0.0071**
			EC3	0.5222	0.7049	0.5254	0.5358	0.5425	0.5321	**0.5230**
10	2^{10}	100	EC1	—	0.0938	0.0820	0.0464	0.0518	0.0746	**0.0461**
			EC2	—	0.2005	0.1751	**0.0266**	0.0383	0.1258	0.0303
			EC3	0.2468	0.4474	0.4261	**0.2666**	0.2798	0.3773	0.2728
		1000	EC1	—	0.1082	0.0434	0.0384	0.0456	0.0419	**0.0288**
			EC2	—	0.2832	0.0952	0.0185	0.0286	0.0687	**0.0152**
			EC3	0.3609	0.6680	0.4720	**0.3728**	0.3938	0.4390	0.3795
		5000	EC1	—	0.1160	0.0316	0.0384	0.0459	0.0293	**0.0218**
			EC2	—	0.3291	0.0592	0.0183	0.0293	0.0442	**0.0089**
			EC3	0.4548	0.8140	0.5294	0.4656	0.4879	0.5041	**0.4621**

Table 1: The average evaluation criteria over the $R = 20$ repetitions of the low-dimensional continuous data simulation studies in Section 4.2.1, based on all coalitions $\mathcal{S} \in \mathcal{P}(\mathcal{M})$.

as reported in Appendix F. We found $K = 250$ to be a suitable trade-off between precision and computation time. However, we use $K = 5000$ to compute the true counterparts.

4.2.1 Continuous Data: Low-Dimensional

In the low-dimensional setting, that is, $M \in \{5, 10\}$, we use all coalitions $\mathcal{S} \in \mathcal{C} = \mathcal{P}(\mathcal{M})$. We consider three different training sample sizes $N_{\text{train}} \in \{100, 1000, 5000\}$, while $N_{\text{test}} = 100$. The smallest training size is chosen to demonstrate that `VAEAC` works well even for limited data, in contrast to the common belief that neural networks need large amounts of data to be properly fitted. We repeat each experiment $R = 20$ times for each approach and report the average evaluation criteria over those repetitions.

The results of the experiments are shown in Table 1, where the best performing method is indicated with bold font for each experiment and evaluation criterion. The `true` approach is only included for EC3, as it is the reference value for the other methods in EC1 and EC2. As previously mentioned, we consider EC1 to be the most informative criterion. We see that `VAEAC` is the overall most accurate method with respect to all criteria. It obtains the lowest EC1 in all experiments, and often with a significant margin, while EC2 and EC3 indicates that the `Gaussian` approach is marginally better when $M = 10$ and $N_{\text{train}} \leq 1000$.

In general, most of the methods become more accurate when we increase N_{train}, except the `independence` approach which becomes less precise. This tendency is most evident

for the `empirical` method, but it is also clear for the `ctree` and `VAEAC` approaches. The `Gaussian` and (Gaussian) `copula` methods do not benefit from increasing N_{train} from 1000 to 5000. That is, the estimates of the parameters in these methods are stable, and the approaches cannot achieve better evaluation scores as they are based on an incorrect parametric assumption about the data distribution.

The `VAEAC` models used in these simulations consist of 17294 ($M = 5$) and 18724 ($M = 10$) parameters, see Appendix C.2, which might sound large considering the small training sets. However, recall from Section 3 that each observation can be masked in 2^M different ways, hence, `VAEAC`'s parameters are fitted based on $2^M \times N_{\text{train}}$ possible different observations. In contrast, the `Gaussian` and `copula` models have both $M(M+3)/2$ parameters each, while the `independence` approach has zero parameters as it randomly samples the training data.

There are some inconsistency between the evaluation criteria. In the $M = 10$ and $N_{\text{train}} = 1000$ setting, the EC3 indicates that the `Gaussian` approach is the best at estimating the contribution functions, with the `VAEAC` method a close second. Furthermore, they are both close to the optimal value. However, when considering the EC1, the `Gaussian` method yields 33% less accurate Shapley value estimates than the `VAEAC` approach, with respect to the mean absolute error in EC1. The inconsistency between EC1 and EC3 can be linked to two things. First, the EC3 evaluates the average precision of the contribution functions, while the EC1 measures the precision of the Shapley values. Second, the estimates of $v(\mathcal{S})$ can overshoot for some coalitions and undershoot for others, and such errors may cancel each other out in the Shapley value formula. Overall, there seems to be a moderate coherency between the criteria, at least in the ranking of the approaches. For real-world data, only the EC3 can be used as the true quantities are unknown. However, one should bear in mind that this criterion can only be used to rank the different approaches and not assess the precision of the estimated Shapley values.

4.2.2 Continuous Data: High-Dimensional

We now let the number of features to be $M \in \{25, 50, 100, 250\}$. In these high-dimensional settings, we sample $N_{\mathcal{S}} = 1000$ coalitions from $\mathcal{P}(\mathcal{M})$ since it is computationally intractable to consider all 2^M coalitions, as described in Section 4.1. Furthermore, we decrease R from 20 to 10 due to increased computational times, see Section 4.4.

The results of the high-dimensional simulation study is presented in Table 2, where there are several noteworthy findings. The `ctree` approach is not applied in the higher dimensions due to increased memory and time consumption. The `VAEAC` method works well for $M = 25$, and is the best approach with regard to EC1 for all training sizes. As expected, when $M \geq 50$, we see that using our $\texttt{VAEAC}_{\mathcal{C}}$ approach, which focuses on the relevant coalitions, greatly increases the performance of the `VAEAC` methodology. Furthermore, we see that $\texttt{VAEAC}_{\mathcal{C}}$ is the overall best approach when N_{train} is sufficiently large. For limited training sizes, the `Gaussian` and `copula` approaches outperform the $\texttt{VAEAC}_{\mathcal{C}}$ method. We have used different values of N_{train} in the different dimensions to illustrate the threshold where $\texttt{VAEAC}_{\mathcal{C}}$ consistently outperforms the other methods. Finally, for $M \in \{50, 100, 250\}$ all three evaluation criteria yield the same ranking of the approaches.

				Methods							
M	$N_{\mathcal{S}}$	N_{train}	Crit	`True`	`Indep.`	`Empir.`	`Gauss.`	`Copula`	`Ctree`	`VAEAC`	$\texttt{VAEAC}_{\mathcal{C}}$
25	1000	100	EC1	—	0.0796	0.0626	0.0523	0.0536	0.0777	**0.0398**	0.0419
			EC2	—	0.4991	0.3124	**0.0643**	0.0718	0.3519	0.1001	0.0663
			EC3	0.2336	0.5947	0.4692	0.2841	0.2913	0.4966	0.3077	**0.2835**
		1000	EC1	—	0.0981	0.0630	0.0368	0.0384	0.0556	**0.0321**	0.0346
			EC2	—	0.7402	0.3010	**0.0468**	0.0570	0.2212	0.0800	0.0496
			EC3	0.3592	0.9022	0.5829	**0.3880**	0.4022	0.5247	0.4164	0.3966
		5000	EC1	—	0.1087	0.0667	0.0357	0.0373	—	**0.0272**	0.0297
			EC2	—	0.8724	0.3101	0.0489	0.0580	—	0.0600	**0.0365**
			EC3	0.4459	1.0775	0.6834	0.4755	0.4886	—	0.4903	**0.4732**
50	1000	5000	EC1	—	0.1669	0.1289	**0.0458**	0.0493	—	0.0531	0.0484
			EC2	—	3.4396	2.1031	**0.1296**	0.1774	—	0.4643	0.1644
			EC3	0.7974	3.7147	2.5857	**0.9087**	0.9585	—	1.2041	0.9396
		10000	EC1	—	0.1710	0.1308	0.0452	0.0495	—	0.0510	**0.0444**
			EC2	—	3.4776	2.0407	0.1261	0.1782	—	0.4281	**0.1203**
			EC3	0.8362	3.8901	2.6502	0.9432	1.0033	—	1.2185	**0.9407**
100	1000	5000	EC1	—	0.2436	0.2281	**0.0555**	0.0566	—	0.1024	0.0594
			EC2	—	6.4815	5.8814	**0.2337**	0.2536	—	1.4657	0.2823
			EC3	0.9556	6.6358	6.0560	**1.1604**	1.1662	—	2.2270	1.2092
		10000	EC1	—	0.2479	0.2359	0.0558	0.0571	—	0.1013	**0.0504**
			EC2	—	6.7383	6.1740	0.2428	0.2653	—	1.4989	**0.1916**
			EC3	1.0333	6.9940	6.4447	1.2458	1.2606	—	2.3644	**1.2097**
250	1000	10000	EC1	—	0.8597	0.9286	0.1612	**0.1467**	—	0.5175	0.1842
			EC2	—	36.4480	43.3528	1.1056	**0.9021**	—	11.9938	1.6597
			EC3	2.5101	37.8259	44.4095	3.6158	**3.3988**	—	14.1812	4.1448
		25000	EC1	—	0.8774	1.0091	0.1658	0.1544	—	0.6891	**0.1474**
			EC2	—	38.2186	48.2050	1.2072	1.0198	—	21.4286	**0.9415**
			EC3	2.8566	39.8862	49.4167	4.1068	3.8659	—	23.8203	**3.7863**

Table 2: The average evaluation criteria over the $R = 10$ repetitions of the high-dimensional continuous data simulation studies in Section 4.2.2, based on the coalitions $\mathcal{S} \in \mathcal{C}$.

We speculate that the performance of our $\texttt{VAEAC}_{\mathcal{C}}$ method could be improved by proper hyperparameter tuning. The same can be said about the `empirical` and `ctree` approaches. We have used the same network architecture for all M, and it might be that, for example, the $d = 8$-dimensional latent space is insufficient for $M = 250$. However, proper hyperparameter tuning of the `VAEAC` methods is computationally expensive and left to further work.

4.3 Simulation Study: Mixed Data

The second type of simulation study concerns mixed data following the setup of Redelmeier et al. (2020), to which we refer for technical details on the data generating process. The essentials are presented in Appendix E.2.

M	M_{cont}	M_{cat}	L	Categorical cut-off values	$\boldsymbol{\beta}_1$	$\boldsymbol{\beta_2}$	$\boldsymbol{\gamma}$
4	2	2	4	$(-\infty, -0.5, 0, 1, \infty)$	1, 0, -1, 0.5	2, 3, -1, -0.5	1, -1
6	4	2	3	$(-\infty, 0, 1, \infty)$	1, 0, -1	2, 3, -0.5	1, -1, 2, -0.25

Table 3: Table of the number of continuous and categorical features, along with the number of categories L and the associated cut-off values for the simulation studies in Section 4.3.1. The last three columns display the true model parameters of (13), while $\alpha = 1$.

4.3.1 Mixed Data: Low-Dimensional

In the exact low-dimensional setting, we consider experiments in dimensions 4 and 6, as outlined in Table 3. Computing the true contribution functions and Shapley values are intractable for larger dimensions due to numerical integration. We investigate several degrees of feature dependencies, and to make the tables more legible, we only report the EC1 (8), which we consider to be the most informative criterion.

To generate M-dimensional dependent mixed data, we first start by generating $N_{\text{train}} \in \{100, 1000, 5000\}$ samples from $\mathcal{N}_M(\mathbf{0}, \Sigma_\rho)$, where the covariance matrix Σ_ρ is the equicorrelation matrix, that is, 1 on the diagonal and ρ off-diagonal. A large ρ implies high feature dependence. The categorical features are created by categorizing the first M_{cat} of the M continuous Gaussian features into L categories at certain cut-off-values, while the remaining M_{cont} continuous features are left untouched, see Table 3. The response is generated according to

$$y = \alpha + \sum_{j=1}^{M_{\text{cat}}} \sum_{l=1}^{L} \beta_{jl} \mathbf{1}(x_j = l) + \sum_{j=M_{\text{cat}}+1}^{M_{\text{cat}}+M_{\text{cont}}} \gamma_j x_j + \epsilon, \tag{13}$$

where the coefficients are defined in Table 3, $\epsilon \overset{iid}{\sim} \mathcal{N}(0, 1)$, and $\mathbf{1}$ is the indicator function. We fit a linear regression model of the same form as (13), but without the error term, to act as the predictive model f. We chose a linear predictive model as the linearity simplifies the computations of the true conditional expectations and, thereby, the true Shapley values for the mixed data setting. The conditional expectations reduce to a linear combination of two types of univariate expectations, which are easy to compute, as shown in Appendix E.2 and Redelmeier et al. (2020). We randomly sample $N_{\text{test}} = 500$ test observations from the joint distribution to compute the average EC1 over R repetitions. The results are presented in Table 4, and we obtain similar results in all experiments.

The `independence` approach is the best method for independent data ($\rho = 0$) regardless of the number of training observations N_{train}. For $\rho = 0.1$, the value of N_{train} determines which method performs the best. The `independence` approach is best for small training sets ($N_{\text{train}} = 100$), `ctree` performs the best for moderate training sets ($N_{\text{train}} = 1000$), and `VAEAC` is the best approach for large training sets ($N_{\text{train}} = 5000$). For moderate to strong dependence ($\rho \geq 0.3$) `VAEAC` is generally the best approach and often with a distinct margin. The `independence` approach performs increasingly worse for increasing dependence, which demonstrates the necessity of methods that can handle dependent data.

M_{cont}	M_{cat}	L	R	N_{train}	Method	$\overline{\text{EC1}}$ for each ρ					
						0.0	0.1	0.3	0.5	0.8	0.9
2	2	4	25	100	`Indep.`	**0.0761**	**0.1310**	0.2364	0.3642	0.5717	0.6540
					`Ctree`	0.0927	0.1412	**0.2048**	0.2442	0.2459	0.2186
					`VAEAC`	0.1338	0.1537	0.2135	**0.1950**	**0.1818**	**0.1949**
				1000	`Indep.`	**0.0289**	0.0811	0.2203	0.3541	0.5615	0.6478
					`Ctree`	0.0320	**0.0706**	0.1050	0.1150	0.1052	0.0954
					`VAEAC`	0.0773	0.0770	**0.0869**	**0.0792**	**0.0667**	**0.0633**
				5000	`Indep.`	**0.0240**	0.0800	0.2163	0.3505	0.5601	0.6464
					`Ctree`	0.0402	0.0603	0.0735	0.0764	0.0685	0.0626
					`VAEAC`	0.0523	**0.0545**	**0.0564**	**0.0555**	**0.0613**	**0.0489**
4	2	3	10	100	`Indep.`	**0.0804**	**0.1180**	0.2649	0.4093	0.6289	0.7196
					`Ctree`	0.0867	0.1327	0.2097	0.2344	0.2084	**0.1814**
					`VAEAC`	0.1606	0.1604	**0.1605**	**0.1840**	**0.1976**	0.1935
				1000	`Indep.`	**0.0312**	0.0924	0.2521	0.4061	0.6382	0.7159
					`Ctree`	0.0352	**0.0801**	0.1110	0.1176	0.1036	0.0844
					`VAEAC`	0.0829	0.0900	**0.0821**	**0.0800**	**0.0704**	**0.0559**
				5000	`Indep.`	**0.0258**	0.0915	0.2551	0.4056	0.6341	0.7074
					`Ctree`	0.0332	0.0619	0.0786	0.0786	0.0676	0.0559
					`VAEAC`	0.0546	**0.0556**	**0.0602**	**0.0553**	**0.0529**	**0.0495**

Table 4: The average EC1 values for the low-dimensional mixed data simulation studies outlined in Table 3 for various dependencies ρ and based on all coalitions $\mathcal{S} \in \mathcal{P}(\mathcal{M})$.

4.3.2 Mixed Data: High-Dimensional

In the high-dimensional mixed data simulation studies, we extend the data generating process described in the low-dimensional setting. We use the EC3 in (10) to rank the approaches, as it is extremely time-consuming to calculate the EC1 and EC2 even in this simple linear simulation setup, see Section 4.4.

We still generate the data according to a $\mathcal{N}_M(\mathbf{0}, \Sigma_\rho)$ and discretize M_{cat} of these features at the categorical cut-off values given in Table 5. We also use the same response function (13) and regression model as in Section 4.3.1. However, to extend the setup such that it works for any M, we sample the model coefficients in (13), while the intercept remains $\alpha = 1$. We sample β_{jl} from $\{-1.0, -0.9, \dots 2.9, 3.0\}$, for $j = 1, \dots, M_{\text{cat}}$ and $l = 1, \dots, L$, and γ_j from $\{-1.0, -0.9, \dots 0.9, 1.0\}$, for $j = 1, \dots, M_{\text{cont}}$. All sampling is done with replacements and equal probability. We only consider $N_{\text{train}} = 1000$, as the other training sizes gave almost identical ranking of the methods in Table 4. The number of test observations and repetitions are $N_{\text{test}} = 500$ and $R = 10$, respectively. Finally, we sample $N_{\mathcal{S}} = 1000$ coalitions in all settings and use 100 epochs to train both the `VAEAC` and $\texttt{VAEAC}_{\mathcal{C}}$ models.

The results of the high-dimensional mixed data studies are presented in Table 6, and they tell a very similar story to that of the low-dimensional simulation studies. For independent data ($\rho = 0$), the `independence` approach is the best method with `ctree` a close second, while the `VAEAC` and $\texttt{VAEAC}_{\mathcal{C}}$ methods falls somewhat behind. For moderate

M	M_{cont}	M_{cat}	L	Categorical cut-off values
10	5	5	3	$(-\infty, 0, 0.5, \infty)$
25	15	10	5	$(-\infty, -1, 0, 0.5, 1, \infty)$
50	25	25	2	$(-\infty, 0.5, \infty)$
50	25	25	4	$(-\infty, -1, 0, 0.5, \infty)$
100	50	50	2	$(-\infty, 0, \infty)$
100	75	25	3	$(-\infty, -1, 1, \infty)$

Table 5: Table of the number of continuous and categorical features, along with the number of categories L, and the associated cut-off values for the high-dimensional mixed data simulation studies.

						$\overline{\text{EC3}}$ for each ρ					
M_{cont}	M_{cat}	L	R	N_{train}	Method	0.0	0.1	0.3	0.5	0.8	0.9
5	5	3	10	1000	`Indep.`	**1.9834**	1.9706	1.9526	1.9940	2.0130	2.0680
					`Ctree`	1.9853	1.9764	1.8568	1.6723	1.1497	0.9079
					`VAEAC`	2.0010	1.9641	**1.7882**	1.5804	1.2869	0.9093
					$\texttt{VAEAC}_{\mathcal{C}}$	2.0032	**1.9610**	1.7893	**1.5721**	**1.2490**	**0.8865**
15	10	5	10	1000	`Indep.`	**4.0698**	4.1725	4.3139	4.4766	4.6167	4.7836
					`Ctree`	4.0740	4.1788	4.0014	3.7119	2.6228	2.0605
					`VAEAC`	4.1396	**4.1592**	3.7626	3.4643	2.5199	2.0980
					$\texttt{VAEAC}_{\mathcal{C}}$	4.1445	4.1886	**3.7195**	**3.4047**	**2.3611**	**1.8218**
25	25	2	10	1000	`Indep.`	**5.1431**	5.5418	4.9215	4.8469	4.6949	4.6823
					`Ctree`	5.1486	5.5480	4.5682	3.8533	2.2973	1.6329
					`VAEAC`	5.3110	5.5235	4.4277	3.6303	2.2686	1.6559
					$\texttt{VAEAC}_{\mathcal{C}}$	5.3121	**5.5137**	**4.2855**	**3.4714**	**2.1214**	**1.3864**
		4	10	1000	`Indep.`	**7.1610**	7.2926	7.6252	7.6523	7.9985	7.9451
					`Ctree`	7.1680	7.3943	7.1921	6.3603	4.5871	3.4432
					`VAEAC`	7.3535	**7.2221**	6.8976	6.0281	4.4857	3.4131
					$\texttt{VAEAC}_{\mathcal{C}}$	7.3556	7.2569	**6.7564**	**5.7906**	**4.0719**	**3.0303**
50	50	2	10	1000	`Indep.`	**10.0998**	10.3732	11.0575	11.6383	12.7133	13.2549
					`Ctree`	10.1195	10.4232	9.8860	8.4248	5.1261	3.7018
					`VAEAC`	10.5142	10.3583	9.2926	8.0861	5.0971	4.3668
					$\texttt{VAEAC}_{\mathcal{C}}$	10.5750	**10.3556**	**8.8561**	**7.1282**	**4.2543**	**2.8744**
75	25	3	10	1000	`Indep.`	**8.2279**	8.2268	7.9975	7.8198	7.5932	7.7030
					`Ctree`	8.2329	**8.2255**	7.4081	6.1508	3.6026	2.4320
					`VAEAC`	8.6734	8.3551	7.3716	6.3859	4.1005	3.0407
					$\texttt{VAEAC}_{\mathcal{C}}$	8.6284	8.2754	**6.8487**	**5.4040**	**2.9895**	**1.9742**

Table 6: The average EC3 values for the high-dimensional mixed data simulation studies outlined in Table 5 for various dependencies ρ and based on the coalitions $\mathcal{S} \in \mathcal{C}$.

to strong dependence ($\rho \geq 0.3$), $\texttt{VAEAC}_{\mathcal{C}}$ is undeniably the best approach, as it significantly outperforms the other methods in all but one setting where it is marginally beaten by `VAEAC`. The margin by which the $\texttt{VAEAC}_{\mathcal{C}}$ outperforms the other methods increases when the dimension and/or the dependence increases. The former follows from $\texttt{VAEAC}_{\mathcal{C}}$'s simple but effective masking scheme, which enables it to focus on the relevant coalitions in $\mathcal{C}$.

			Methods						
M	$N_{\mathcal{S}}$	N_{train}	`Indep.`	`Empir.`	`Gauss.`	`Copula`	`Ctree`	$\texttt{VAEAC}_{\mathcal{C}}$	
5	2^5	100	0.4	0.4	0.6	0.9	0.3	1.1	(0.2)
		1000	1.1	1.1	0.9	1.2	0.6	2.1	(1.0)
		5000	1.5	1.5	1.1	1.5	1.0	4.2	(2.9)
10	2^{10}	100	11.5	11.5	25.9	44.8	23.0	24.2	(0.3)
		1000	34.8	35.0	36.4	65.4	45.8	39.7	(1.3)
		5000	43.5	44.1	49.0	88.3	121.2	39.7	(4.3)
25	1000	100	8.5	8.5	17.4	38.3	18.4	28.8	(2.4)
		1000	27.0	27.0	21.6	45.7	57.6	41.8	(11.0)
		5000	38.2	36.8	24.5	56.2	—	65.2	(32.2)
50	1000	5000	65.1	62.4	32.1	76.9	—	101.6	(54.7)
		10000	122.3	121.8	69.4	96.1	—	165.4	(78.4)
100	1000	5000	171.5	167.2	48.4	112.2	—	167.2	(92.5)
		10000	210.1	208.7	86.0	197.0	—	428.3	(326.7)
250	1000	10000	380.0	466.6	97.8	192.1	—	433.2	(311.9)
		25000	901.8	992.1	79.6	232.2	—	614.7	(487.8)

Table 7: Average CPU times in minutes needed to compute the Shapley values using the different methods for $N_{\text{test}} = 100$ observations in the continuous simulation studies in Section 4.2.

4.4 Computation Times

The evaluation of the methods should not be limited to their accuracy, but also include their computational complexity. The CPU times of the different methods will vary significantly depending on operating system and hardware. We ran the simulations 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. The `VAEAC` and $\texttt{VAEAC}_{\mathcal{C}}$ approaches are identical, except in the masking scheme, and therefore obtain nearly identical running times. We only report the time of $\texttt{VAEAC}_{\mathcal{C}}$ and rather include the training time of the model in parenthesis.

In Table 7, we display the average CPU times in minutes needed for each approach to explain $N_{\text{test}} = 100$ test observations, averaged over the R repetitions, for the continuous data simulation study in Section 4.2. We see that the competing methods are generally faster than the $\texttt{VAEAC}_{\mathcal{C}}$ approach, except for the `ctree` method for larger training sizes. When $N_{\text{train}} = 100$, the `independence` and `empirical` approaches are the fastest methods. For larger training sizes ($N_{\text{train}} \geq 1000$) and higher dimensions ($M \geq 25$), the `Gaussian` approach is the fastest method. The CPU times of the `independence` approach are higher than expected for large M. This is probably due to the fact that the `independence` approach is implemented as a special case of the `empirical` approach in version 0.2.0 of the `shapr`-package.

When N_{train} increases, most methods yield larger computational times. $\texttt{VAEAC}_{\mathcal{C}}$'s time increase is mainly caused by the training phase and not by the deployment phase in the

					Methods			
M_{cont}	M_{cat}	L	$N_{\mathcal{S}}$	N_{train}	`Indep.`	`Ctree`	$\texttt{VAEAC}_{\mathcal{C}}$	
2	2	4	2^4	100	0.1	0.6	1.8	(0.1)
				1000	0.4	0.6	2.3	(0.5)
				5000	0.5	0.7	4.1	(2.4)
4	2	3	2^6	100	0.2	2.9	2.2	(0.1)
				1000	1.2	2.9	2.7	(0.6)
				5000	1.6	3.0	5.1	(3.0)
5	5	3	1000	1000	4.0	30.8	18.9	(0.9)
15	10	5	1000	1000	13.5	258.3	73.8	(3.6)
25	25	2	1000	1000	37.8	273.2	156.8	(5.4)
		4	1000	1000	39.7	1326.4	141.2	(6.4)
50	50	2	1000	1000	98.8	2385.2	276.4	(10.1)
75	25	3	1000	1000	68.1	2453.8	248.8	(10.2)

Table 8: Average CPU times in minutes needed to compute the Shapley values using the different methods for $N_{\text{test}} = 500$ observations in the mixed data simulation studies in Section 4.3.

continuous data simulation studies. When we train the NNs in $\texttt{VAEAC}_{\mathcal{C}}$, we have used a conservative fixed number of epochs, see Appendix C.2. Thus, when N_{train} increases, each epoch takes longer time to run. For example, when $M = 250$ and $N_{\text{train}} = 25000$, the training phase of the $\texttt{VAEAC}_{\mathcal{C}}$ method constitutes 79% of the total running time. Optimal choice of hyperparameters (epochs, learning rate, NN architecture) and use of early stopping regimes might decrease the training time without loss of accuracy. However, the training time is a fixed time-cost, which would be near negligible if we were to explain, for example, $N_{\text{test}} = 10^6$ observations instead of only $N_{\text{test}} = 100$.

In Table 8, we display the average CPU times in minutes needed for each approach to explain $N_{\text{test}} = 500$ test observations for the mixed data simulation studies in Section 4.3. We see that the `independence` approach is the fastest method in all settings. However, the $\texttt{VAEAC}_{\mathcal{C}}$ approach is much faster than the `ctree` method for $M = M_{\text{cat}} + M_{\text{cont}} \geq 10$. As $N_{\text{train}} = 1000$ in all high-dimensional settings ($M \geq 10$), the increase in CPU time for $\texttt{VAEAC}_{\mathcal{C}}$ is mainly caused by increased sampling time in the deployment phase and not the training time in the training phase, which slowly increases with M.

The true Shapley values took 51.93 minutes to compute in the $M = 4$-dimensional experiments and 30.33 hours to compute in the $M = 6$ setting, on average, for one repetition of one value of ρ. Thus, in total it took 243.70 CPU days to compute the true Shapley values needed to compute the results in Table 4. This illustrates why computing EC1 and EC2 is computationally infeasible for the high-dimensional mixed data simulation studies, hence, why we only used the EC3 criterion to rank the approaches in Table 6.

5. Application on Real Data Example

In this section, we explain predictions made by a random forest model fitted to 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). It has been used in several XAI papers (Vilone et al., 2020; Aas et al., 2021b; Frye et al., 2021) and other machine-learning studies (Sahin et al., 2018; Smith et al., 2018; Mohammed et al., 2020) as it is freely available from the UCI Machine Learning Repository `http://archive.ics.uci.edu/ml/datasets/Abalone`.

An abalone is an edible marine snail, and the harvest of abalones is subject to quotas that are partly based on the age distribution. It is a time-consuming task to determine the age of abalones, as one has to cut the shell through its cone, stain it, and manually count the number of rings through a microscope. The goal is therefore to predict the age of abalones based on other easier obtainable physical measurements. The Abalone data set consists of 4177 samples with 9 features each: `Rings` (+1.5 gives the age in years), `Length` (longest shell measurement), `Diameter` (perpendicular to length), `Height` (with meat in shell), `WholeWeight` (whole abalone), `ShuckedWeight` (weight of meat), `VisceraWeight` (gut-weight after bleeding), `ShellWeight` (after being dried), and `Sex` (female, infant, male). The distances are given in millimeters and weights in grams. `Rings` is an integer, `Sex` is categorical with 3 categories, and the rest of the features are continuous. Hence, `VAEAC` is a suitable method for this mixed data set, in addition to the other methods in Section 4.3.

An overview of the Abalone data set can be seen in Figure 3, where we show the pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients (for continuous features). There is a clear non-linearity and heteroscedasticity among the pairs of features, and there is significant pairwise correlation between the features. All continuous features have a pairwise correlation above 0.775, or 0.531 when grouped by `Sex`.

Aas et al. (2021b) also used the Abalone data set when demonstrating their method for estimating Shapley values. However, their method lacks support for categorical features. Hence, they discarded the categorical feature `Sex`. We consider this to be a critical methodological limitation, as we will show that `Sex` is the most important feature for some test observations. This is not surprising, as Figure 3 displays a clear distinction between infants and females/males. Hence, it is important to include `Sex` in the prediction problem.

We treat the `Rings`/age prediction as a regression problem. We split the data set at random into a test and training set consisting of 100 and 4077 samples, respectively. We use a random forest model as it can detect any potential non-linear relationships between the response and the features. The model was fitted to the training data using the R-package `ranger` (Wright and Ziegler, 2017) with 500 trees and default parameter settings. In a real use case, one would typically not directly use an off-the-shelf ML model, but rather consider, for example, possible feature transformations and conduct model selection and hyper-parameter tuning before using the model. However, as this section aims to illustrate the proposed `VAEAC` approach and the local Shapley value methodology on a practical example with real-world data, we have taken the liberty to use the default version of random forest in `ranger`.

Figure 4 shows the estimated local Shapley values for three test observations, based on the `independence`, `ctree`, and `VAEAC` approaches. Recall from Section 2.2 that the Shapley

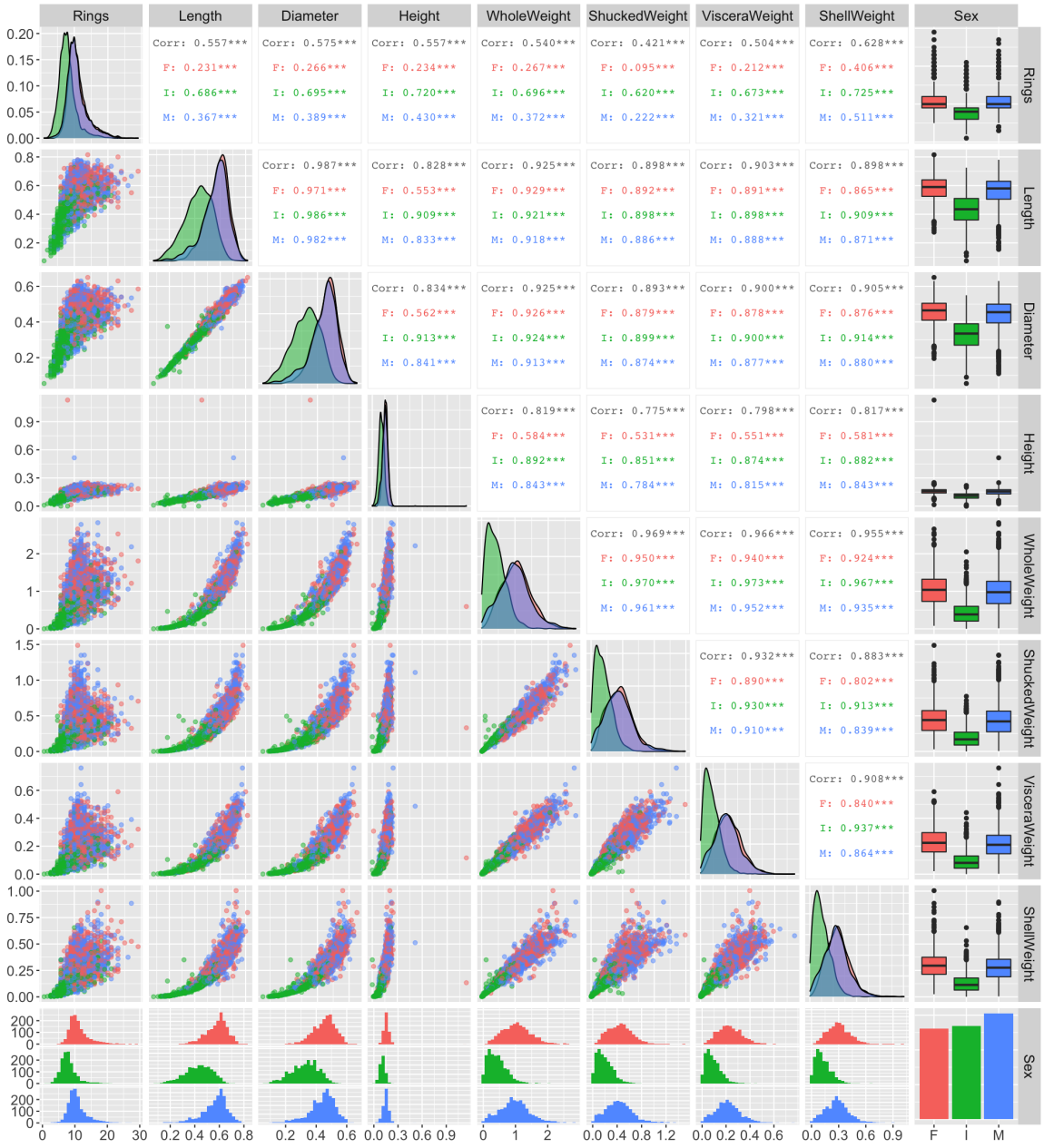

Figure 3: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the response `Rings` and the features of the Abalone data set. The figure is grouped by `Sex`, where the females are red, infants are green, and males are blue. The correlations reported in black correspond to all observations, while the colored correlations are grouped based on `Sex`. The data is highly correlated and display a clear distinction between infants and females/males.

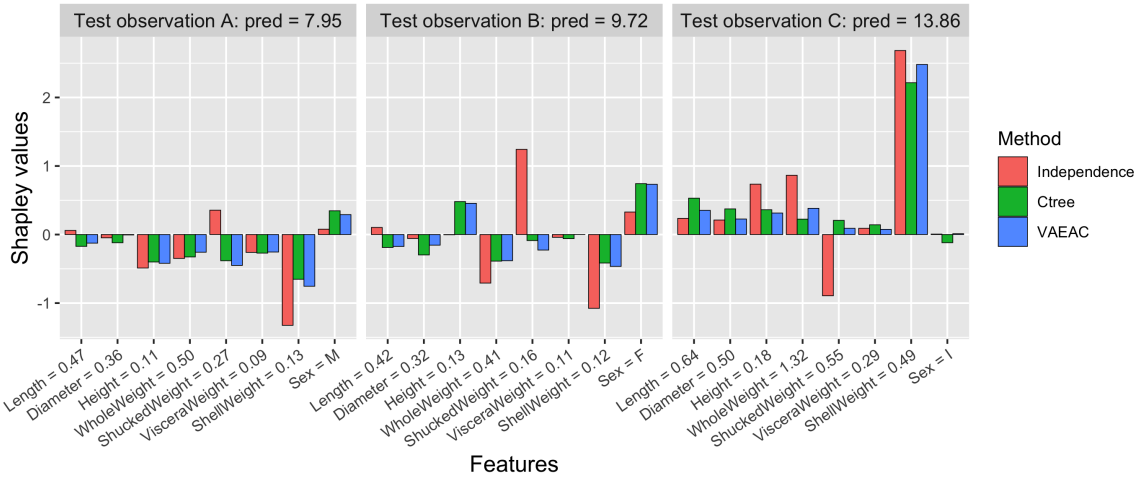

Figure 4: Shapley values for three test observations, where the predicted response is included in the header. The `VAEAC` and `ctree` approaches yield similar estimates, while the `independence` approach often differ in both magnitude and direction.

values for a test observation $\boldsymbol{x}^*$ explain the difference between the prediction $f(\boldsymbol{x}^*)$ and the global average prediction $\phi_0 = \mathbb{E}[f(\boldsymbol{x})]$. We estimate ϕ_0 by the mean of the response in the training set, which is $\phi_0 = 9.93$ in our training set. The three displayed test observations are chosen such that their predicted responses are, respectively, below, similar to, and above ϕ_0. Moreover, they have different values of the feature `Sex`. This entails that observation A has mostly negative Shapley values, while observation B has a mixture of positive and negative Shapley values, and observation C has primarily positive Shapley values.

The Shapley values based on the `ctree` and `VAEAC` approaches are comparable. For both, `Sex` gets the largest absolute Shapley value for test observation B. That is, this categorical feature is regarded to be the most important feature for explaining the prediction for this test observation. The importance of `Sex` is similar to that of other features for test observation A, while it is negligible for test observation C. For test observations A and C, `ShellWeight` is the most important feature.

The `independence` approach often yields distinctly different results than the `ctree` and `VAEAC` methods, both in magnitude and direction. We do not trust the Shapley values of the `independence` approach, as there is strong feature dependence in our data. Based on the poor performance of the `independence` approach for dependent data in the simulation studies, we strongly believe the associated explanations to be incorrect and that they should be discarded. In the next section, we further justify the use of the `ctree` and `VAEAC` approaches.

5.1 Evaluation

For all approaches treated in this paper, the Shapley value in (1) is a weighted sum of differences $v(\mathcal{S}\cup\{j\}) - v(\mathcal{S})$ over possible subsets $\mathcal{S}$. However, the approaches differ in how $v(\mathcal{S})$, or more specifically, the conditional distribution $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$, is estimated. We now illustrate that the conditional samples generated using the `VAEAC` and `ctree` approaches are more representative than the samples generated using the `independence` method. Thus,

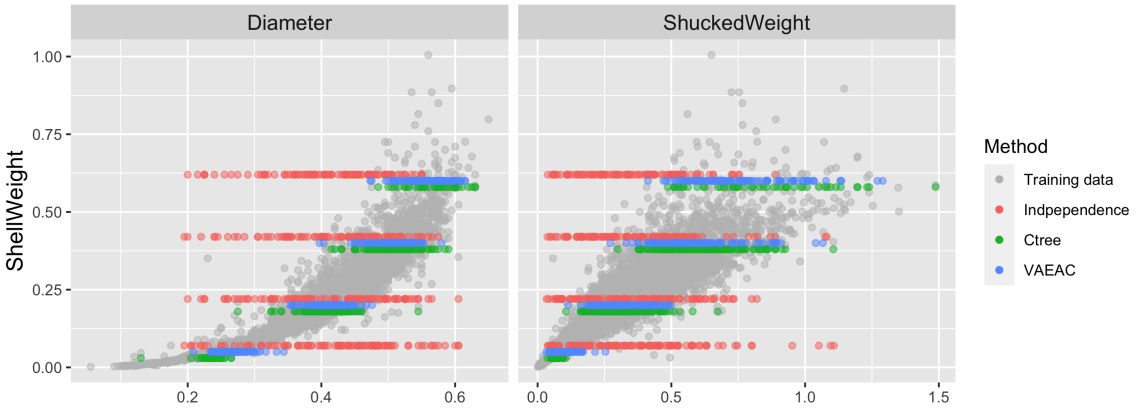

Figure 5: Sampling from the estimated conditional distributions using the `independence` (red), `ctree` (green), and `VAEAC` (blue) approaches. The gray dots are the training data. The samples are generated conditioned on `ShellWeight` $\in \{0.05, 0.2, 0.4, 0.6\}$. Note that the red and green dots have been slightly displaced vertically to improve visibility of the figure. The `ctree` and `VAEAC` approaches yield on-distribution data, while the `independence` method generates samples off-distribution.

it is likely that the estimated Shapley values using the former two methods also are more accurate for the Abalone example.

Since there are $2^M = 2^8 = 256$ conditional distributions involved in the Shapley formula, it is impossible to show all here. However, we have included some examples to illustrate that the `VAEAC` and `ctree` approaches give more correct approximations to the true conditional distributions than the `independence` approach. Figure 5 shows pairplots of `ShellWeight` against both `Diameter` and `ShuckedWeight`, where the grey dots are the training data. The red, green, and blue dots are generated samples from the estimated conditional distribution of the feature on the x-axis given `ShellWeight` $\in \{0.05, 0.2, 0.4, 0.6\}$, using the `independence`, `ctree`, and `VAEAC` approaches, respectively.

The `independence` approach generates unrealistic samples far outside the range of observed values in the training data, in contrast to the `VAEAC` and `ctree` methods. It is well known that evaluation of predictive machine learning models far from the domain at which they have been trained, can lead to spurious predictions (Nguyen et al., 2015; Goodfellow et al., 2015), which is related to the *garbage-in-garbage-out* problem. Thus, it is important that the explanation methods are evaluating the predictive model at appropriate feature combinations.

We can also use the EC3, defined in (10), to rank the approaches. Recall that the EC3 can be used when the true data generating process is unknown. However, we can only rank the approaches and not state how close they are the optimal approach. Based on the discussion above and Figure 5, it is natural to expect the `independence` approach to obtain a high value relative to the `ctree` and `VAEAC` approaches, which should obtain similar low values. The `independence` approach produces a value of 2.955, while the `ctree` and `VAEAC` methods obtain 0.995 and 0.942, respectively. Thus, according to the EC3, the most accurate approach is `VAEAC`.

6. Summary and Conclusion

Shapley values originated in cooperative game theory as a solution concept of how to fairly divide the payout of a game onto the players and have a solid theoretical foundation therein. It gained momentum as a model-agnostic explanation framework for explaining individual predictions following the work of Lundberg and Lee (2017). However, their work relied on the assumption of independent features. Thus, their methodology may lead to incorrect explanations when the features are dependent, which is often the case for real-world data. There have been several proposed methods for appropriately modeling the dependence between the features, but this is computationally expensive as the number of feature combinations to model grows exponentially with the number of features.

We introduce the variational autoencoder with arbitrary conditioning `VAEAC` (Ivanov et al., 2019) as a tool to estimate the conditional expectations in the Shapley value explanation framework. The `VAEAC` approach uses a single variational autoencoder to simultaneously estimate the dependence structure between all feature combinations. This is in contrast to the state-of-the-art dependence-aware methods available in the R-package `shapr` (Sellereite and Jullum, 2019), which fits a new model to each feature combination. Furthermore, `VAEAC` supports a mixture of continuous and categorical features. The `ctree` approach (Redelmeier et al., 2020) is the only other method that directly supports dependent mixed features. The other approaches could be extended to support categorical features by using *external* one-hot encoding, but this is in general infeasible as it drastically increases the computational complexity (Redelmeier et al., 2020, q.v. Table. 4 & 6), in particular for categorical features with many categories.

Through a series of low-dimensional simulation studies, we demonstrated that our `VAEAC` approach to Shapley value estimation outperforms or matches the state-of-the-art dependence-aware approaches, even for very small training sets. In the high-dimensional simulation studies, we used efficient sampling of coalitions and evaluated the approaches based on the sampled coalitions. Evaluating all coalitions is infeasible in high dimensions as the number of feature dependencies to estimate grows exponentially with the number of features. We introduced our $\texttt{VAEAC}_{\mathcal{C}}$ approach, which employs a simple but effective non-uniform masking scheme that allows the $\texttt{VAEAC}_{\mathcal{C}}$ approach to focus on the relevant coalitions. This significantly improved the accuracy of the `VAEAC` methodology in the high-dimensional simulation studies, as our novel $\texttt{VAEAC}_{\mathcal{C}}$ approach outperformed all other methods by a significant margin when trained on sufficiently large training sets. Furthermore, our $\texttt{VAEAC}_{\mathcal{C}}$ approach was also computationally competitive with the other methods. Further work on proper hyper-parameter tuning of the $\texttt{VAEAC}_{\mathcal{C}}$ method might decrease computation times.

Another possible application of masking schemes is in the *asymmetric Shapley value* framework of Frye et al. (2020), which under certain conditions coincides with the methodology of Heskes et al. (2020, Corollary 3). The asymmetric Shapley value incorporates prior knowledge of the data's casual structure (should such be known) into the explanations of the model's predictions. In their framework, for distal/root causes, it is natural to consider a masking scheme that solely focuses on coalitions where the ancestor features are known, and the descendant features are unknown. For proximate/immediate causes, the reverse masking scheme would be suitable.

The `VAEAC` approach was also used to explain predictions made by a random forest model designed to predict the age of an abalone (marine snail). In this case, the true Shapley values are unknown, but we saw a strong coherency between the explanations from the `VAEAC` and `ctree` approaches. The `independence` approach gave different Shapley values both in direction and magnitude. We provided results that indicate that the former two approaches provide more sensible Shapley value estimates than the latter. Furthermore, our `VAEAC` approach was the best method according to the evaluation criterion.

The focus of this article has been on modeling the feature dependencies in the *local* Shapley value framework. However, these approaches can also be used to model feature dependencies in *global* Shapley value frameworks. For example, Covert et al. (2020) currently assume feature independence when estimating their global Shapley values. Thus, their framework would benefit from using, e.g., the `VAEAC` approach.

In the current implementation of `VAEAC`, we use the Gaussian distribution to model the continuous features. This can generate inappropriate values if the feature values are, for example, strictly positive or restricted to an interval (a, b). In such settings `VAEAC` can generate negative samples or samples outside the interval, respectively. A naive approach is to set the negative values to zero and the values outside the interval to the nearest endpoint value. However, a better approach would be to transform the data to an unbounded form before using the `VAEAC` approach. That is, strictly positive values can be log-transformed, while interval values can first be mapped to $(0, 1)$ and then further to the full real line by, for example, the logit function. A more complicated but undoubtedly interesting approach is to replace the Gaussian distribution in the decoder with a distribution that has support on the same range as the features, like the gamma distribution for strictly positive values and the beta distribution for values in $(0, 1)$.

The `VAEAC` approach currently supports incomplete data in the training phase, but the methodology can also be extended to support incomplete data in the deployment phase. This could be done by including a missing feature mask in the masked encoder, similar to that we explained for the full encoder in Section 3.5. This would allow the `VAEAC` method to condition on missing values. That is, in the deployment phase, the masked encoder would receive the observed features $\boldsymbol{x}_{\mathcal{S}}$ of the test sample $\boldsymbol{x}$, with missing values set to zero, the unobserved feature mask $\bar{\mathcal{S}}$, and the missing feature mask indicating the missing features. The missing feature mask is needed to distinguish between actual zeros in $\boldsymbol{x}_{\mathcal{S}}$ and zeros induced by us setting missing features to zero. We discuss a small example in Appendix C.4.

Acknowledgments

This work was supported by The Norwegian Research Council 237718 through the Big Insight Center for research-driven innovation. We thank Annabelle Redelmeier for her advice on setting up the low-dimensional mixed data simulation study. Furthermore, we are immensely grateful to the anonymous reviewers for their suggestions which highly improved the manuscript.

Appendix

We start the appendix by giving a brief description of the alternative approaches to `VAEAC` in Appendix A. In Appendix B, we give the full derivation of the variational lower bound. The implementation details of `VAEAC` are discussed in Appendix C. In Appendix D, we conduct a simulation study with only categorical features. The data generating process of the different simulation studies are elaborated in Appendix E. In Appendix F, we experiment with different number of Monte Carlo samples on the MAE. Finally, Appendix G illustrates `VAEAC` as a universal approximator on the Abalone data set.

Appendix A. Alternative Approaches

In this section, we give a short description of the other approaches available in the R-package `shapr` (Sellereite and Jullum, 2019), that is, the `independence`, `empirical`, `Gaussian`, `copula`, and `ctree` approaches. We use the default hyperparameters implemented in the `shapr`-package when using these methods.

Method	Citation	Description
`Independence`	Lundberg and Lee (2017)	Assume the features are independent. Estimate (2) by (3) where $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ are subsamples from the training data.
`Empirical`	Aas et al. (2021a)	Calculate the Mahalanobis distance between the observation being explained and every training instance. Use this distance to calculate a weight for each training instance using $\eta = 1$. Approximate (2) using a function of these weights.
`Gaussian`	Aas et al. (2021a)	Assume the features are jointly Gaussian. Sample K times from the corresponding conditional distribution. Estimate (2) with (3) using these samples.
`Copula`	Aas et al. (2021a)	Assume the dependence structure of the features can be approximated by a Gaussian copula. Sample K times from the corresponding conditional distribution. Estimate (2) with (3) using these samples.
`Ctree`	Redelmeier et al. (2020)	Fit conditional inference trees for each coalition. Sample K times from the corresponding conditional distribution (with replacements). Estimate (2) with (3) using these samples.

Table 9: A short description of the approaches used to estimate (2) in the simulation studies.

Appendix B. Variational Lower Bound for VAEAC

Here we give the full derivation of the variational lower bound in (6):

$$
\begin{aligned}
\log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}}) &= \int p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}}) \log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\,\mathrm{d}\boldsymbol{z} \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\underbrace{\boldsymbol{x}_{\mathcal{S}},\boldsymbol{x}_{\bar{\mathcal{S}}}}_{\boldsymbol{x}},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \left\{\frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\frac{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right\}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] + \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] + \underbrace{D_{\mathrm{KL}}\big(p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}}) \parallel p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})\big)}_{\geq 0} \\
&\geq \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}},\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\right] + \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\right] - \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log \frac{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}{p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})}\right] \\
&= \mathbb{E}_{p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z},\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\right] - D_{\mathrm{KL}}\big(p_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x},\bar{\mathcal{S}}) \parallel p_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}_{\mathcal{S}},\bar{\mathcal{S}})\big) \\
&= \mathcal{L}_{\texttt{VAEAC}}(\boldsymbol{x},\bar{\mathcal{S}}|\boldsymbol{\theta},\boldsymbol{\psi},\boldsymbol{\varphi}).
\end{aligned}
$$

Appendix C. Implementation Details of VAEAC

In this section, we describe different implementation details, masking schemes, and possible implementation improvements for the VAEAC approach.

C.1 Fixed-Length Input and Output

In Section 3, we used a simplified representation to focus the spotlight on the essential aspects of the VAEAC approach. We now provide the full implementation details.

The VAEAC method is implemented in the auto-grad framework PyTorch (Paszke et al., 2019). In the implementation, we need to take into consideration that the number of unobserved features $|\bar{\mathcal{S}}|$ varies depending on the coalition $\mathcal{S}$. Hence, the input of the full encoder and masked encoder will be of varying length, which is inconsistent with NNs' assumption of fixed-length input. Figure 6 is an extended version of Figure 2 where we include the encoding layer L^* in the encoders, which one-hot encodes the categorical features

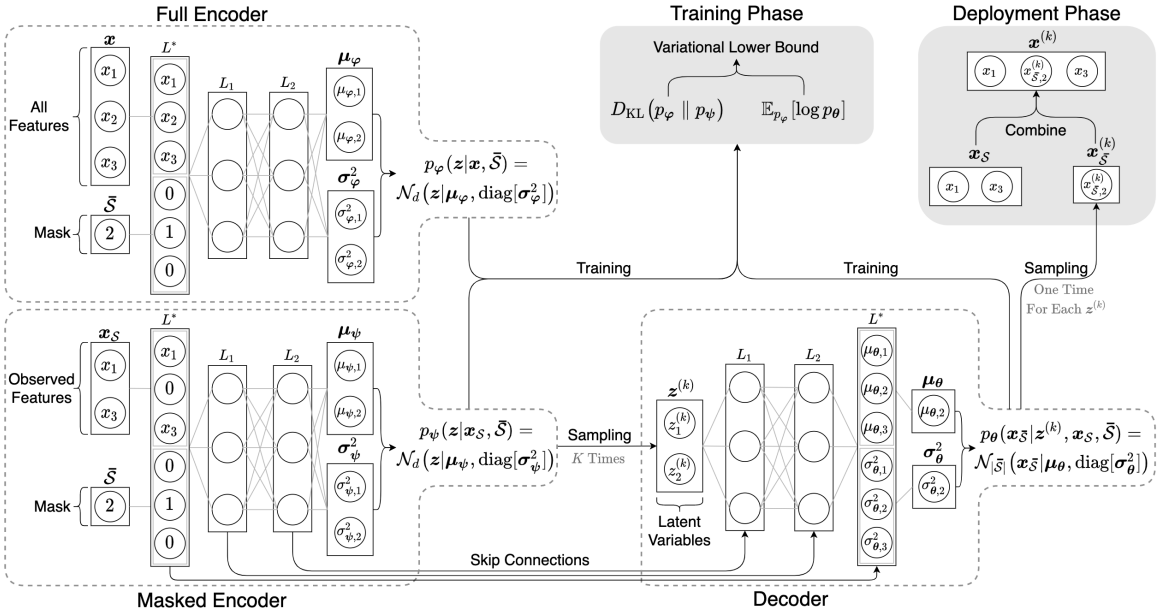

Figure 6: An extended version of Figure 2 where we include the fixed-length implementation details introduced in Appendix C.1.

and enforces a fixed-length input to the first layer L_1 of the NNs. To ensure masks $\bar{\mathcal{S}}$ of equal length, we introduce $I(\bar{\mathcal{S}}) = \left\{\mathbf{1}(j \in \bar{\mathcal{S}}) : j = 1, \ldots, M\right\} \in \{0,1\}^M$, which is an M-dimensional binary vector where the jth element $I(\bar{\mathcal{S}})_j$ is 1 if the jth feature is unobserved and 0 otherwise. For a fixed-length observed feature vector $\boldsymbol{x}_{\mathcal{S}}$, we define $\tilde{\boldsymbol{x}}_{\mathcal{S}} = \boldsymbol{x} \circ I(\mathcal{S})$, where $\circ$ is the element-wise product. Similarly, the decoder also has a fixed-length output. Thus, we also introduce an extraction layer L^* in the decoder, which extracts the relevant values of $\boldsymbol{\mu_\theta}$ and $\boldsymbol{\sigma^2_\theta}$ given mask $\bar{\mathcal{S}}$.

If we did not provide the full encoder with the mask $\bar{\mathcal{S}}$, it would map the input $\boldsymbol{x}$ to the same $\boldsymbol{\mu_\varphi}$ and $\boldsymbol{\sigma^2_\varphi}$ for all $\mathcal{S}$. By also providing $\bar{\mathcal{S}}$, the full encoder can find different latent representations of $\boldsymbol{x}$ for each $\bar{\mathcal{S}}$. The conceptual aim of the full encoder is to guide the masked encoder, in the training phase, to find relevant latent representations of $\boldsymbol{x}_{\mathcal{S}}$ and $\bar{\mathcal{S}}$. Thus, including the mask $\bar{\mathcal{S}}$ in the full encoder will aid this procedure.

We now elaborate the example given in Section 3.2 with $\boldsymbol{x} = \{x_1, x_2, x_3\}$, $\mathcal{S} = \{1, 3\}$ and $\bar{\mathcal{S}} = \{2\}$. The observed and unobserved feature vectors are $\boldsymbol{x}_{\mathcal{S}} = \{x_1, x_3\}$ and $\boldsymbol{x}_{\bar{\mathcal{S}}} = \{x_2\}$, respectively. The associated fixed-length versions computed in the encoding layer L^* are $I(\bar{\mathcal{S}}) = \{0, 1, 0\}$ and $\tilde{\boldsymbol{x}}_{\mathcal{S}} = \{x_1, x_2, x_3\} \circ \{1, 0, 1\} = \{x_1, 0, x_3\}$. Note that `VAEAC` does not have issues differentiating actual zeros from zeros induced by the fixed-length transformation as it has access to $I(\bar{\mathcal{S}})$. We consider these implementation details part of the NN architecture; hence, we have not included the tilde-notation and the I function in the mathematical derivations. The final complete sample is therefore $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k)}_{\bar{\mathcal{S}}} \circ I(\bar{\mathcal{S}}) + \boldsymbol{x} \circ I(\mathcal{S})$.

Figure 7 illustrates the `VAEAC` method's *internal* use of one-hot encoding in the encoding layer L^* for an $M = 4$-dimensional mixed data setting. Assume that features x_1 and x_2 are continuous, while x_3 and x_4 are categorical with 3 categories each. Figure 7 displays the

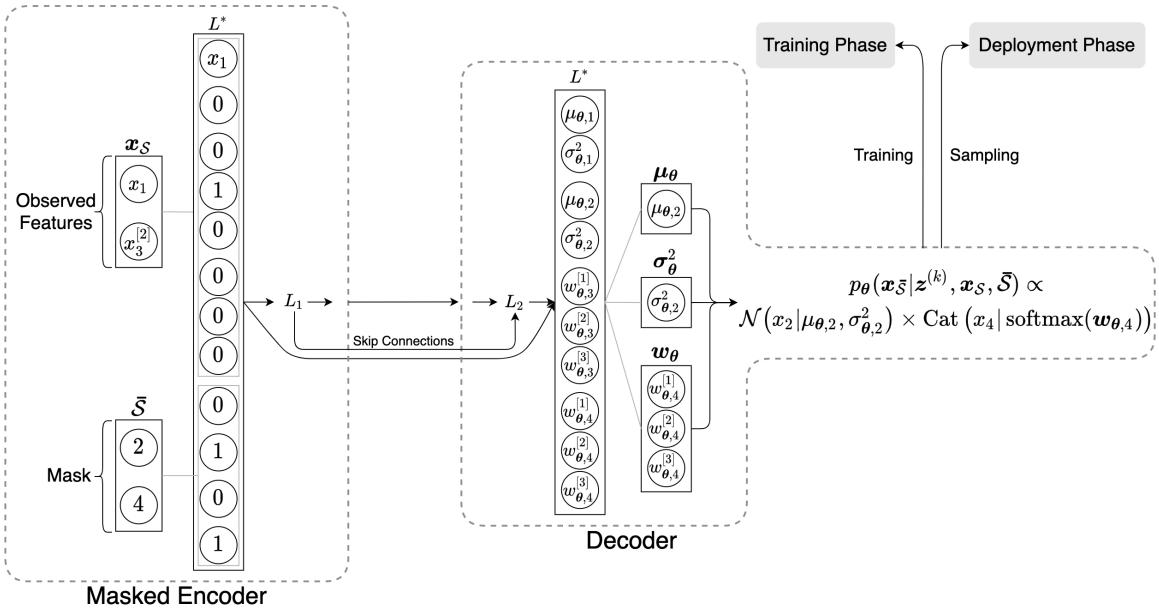

Figure 7: Illustration of `VAEAC` with mixed features and fixed-length input. Features x_1 and x_2 are continuous, while x_3 and x_4 are categorical with 3 categories each. Here $\boldsymbol{x}_{\mathcal{S}} = \{x_1, x_3\}$ and $\boldsymbol{x}_{\bar{\mathcal{S}}} = \{x_2, x_4\}$, i.e., $\bar{\mathcal{S}} = \{2, 4\}$. Furthermore, x_3 takes on the second category indicated by the superscript in $x_3^{[2]}$. The $\boldsymbol{w}_{\boldsymbol{\theta}}$ are the logits of the different categories. The structure of the full encoder is identical to the masked encoder, except that it receives all features during the training phase.

setting where $\bar{\mathcal{S}} = \{2, 4\}$, which means that $\boldsymbol{x}_{\bar{\mathcal{S}}} = \{x_2, x_4\}$ are the unobserved features to be estimated conditioned on the observed features $\boldsymbol{x}_{\mathcal{S}} = \{x_1, x_3\}$. Let x_3 belong to the second category, which we indicate by $x_3^{[2]}$. Thus, the one-hot encoding of $x_3^{[2]}$ is $\{0, 1, 0\}$, while x_4 becomes $\{0, 0, 0\}$ in the one-hot fixed-length input notation generated in the encoding layer L^*. The decoder computes the estimated means $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and variances $\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}$ for the continuous features and logits $\boldsymbol{w}_{\boldsymbol{\theta}}$ for each category of the categorical features. The relevant parameters are extracted by the extraction layer L^* in the decoder.

C.2 Hyperparameters

We must set several hyperparameters before training a `VAEAC` model, both architecturally and training-wise. The NNs of the encoder, masked encoder, and decoder can be of different sizes, but throughout the article, we use shared parameters and set `depth` $= 3$, `width` $= 32$, $d =$ `latent_dim` $= 8$ based on experience. Optimizing these hyperparameters can yield even better results. The number of model parameters N_{param} grows in line with the number of features. For the continuous data simulation studies (Section 4.2), $N_{\text{param}} \in \{17294, 18724, 24214, 37364, 78664, 322564\}$ for $M \in \{5, 10, 25, 50, 100, 250\}$, respectively. For the mixed data simulation studies (Section 4.3), we have that $N_{\text{param}} \in \{17652, 17966, 19879, 32464, 41464, 60064, 91864, 98939\}$ for the eight experiments, in the

order they were presented. We tune the trainable model parameters of the `VAEAC` approach using the Adam optimization algorithm (Kingma and Ba, 2015), with a learning rate `lr` $= 0.001$.

We use the same values for the hyperparameters σ_μ and σ_σ of the prior in latent space as Ivanov et al. (2019), that is, 10^4 for both of them. This value correspond to a very weak, almost disappearing, regularization. That is, the distribution is close to uniform near zero and does not affect the learning process significantly. The activation function used is LeakyReLU, with the default parameters set in the PyTorch implementation. We use `batch_size` $= 64$ and generate $K = 250$ Monte Carlo samples from each conditional distribution for each test observation. The masking scheme $p(\bar{\mathcal{S}})$ is set to have equal probability for all coalitions $\bar{\mathcal{S}}$ for `VAEAC`, while the masking scheme for `VAEAC`$_{\mathcal{C}}$ is the sampling frequencies of the coalitions $\mathcal{S}$ in $\mathcal{C}$, see Section 3.4.

In all simulation studies, we let the number of epochs be `epochs` $= 200$ if $N_{\text{train}} \leq 1000$ and `epochs` $= 100$ if $N_{\text{train}} \geq 5000$, unless otherwise specified. These are conservative numbers. We use the `VAEAC` model at the epoch with the lowest *importance sampling* estimate, denoted by IWAE (Sohn et al., 2015). The IWAE estimates the log-likelihood by

$$\text{IWAE} = \log \frac{1}{V} \sum_{i=1}^{V} \frac{p_{\boldsymbol{\psi}}(\boldsymbol{z}_i|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}) p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{z}_i, \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})}{p_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})} \approx \log p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}}), \tag{14}$$

where V is the number of IWAE validation samples $\boldsymbol{z}_i \sim p_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$ generated for each $\boldsymbol{x}_{\mathcal{S}}$. We set $V = 40$. The IWAE is computed on a hold out validation set which is set to be 25% of the training data. That is, if $N_{\text{train}} = 100$, we actually only use 75 observations to train the `VAEAC` model and the rest to validate the trained model. In our experience, the VLB and IWAE stabilize long before the last epoch, hence, we have chose conservative values of `epochs` which could be optimzed to decrease the training time.

To stabilize the results, we initiate 15 `VAEAC` models, let them run for 5 epochs, and then only continue training the `VAEAC` with lowest VLB. This is to reduce the effect of the randomly initiated model parameters, which can put `VAEAC` in a sub-optimal part of the parameter space which it cannot escape. One could also use other types of adaptive learning rates to solve this problem, see Goodfellow et al. (2016, Ch. 8).

C.3 Improvements in Implementation

The current implementation of `VAEAC` can be improved in several ways. First, we have used a conservative number of epochs during training, but the IWAE (14) often stabilizes before the reaching the last epoch. Hence, the fitting time of `VAEAC` could be reduced by applying some type of early stopping regime if no improvement is seen in the IWAE over several epochs. Second, the sampling process of `VAEAC` is not parallelized. However, as the test observations are independent, the associated conditional samples could be generated simultaneously on separate CPUs or, even better, GPUs. We did not have access to GPUs when we ran the simulations.

C.4 Incomplete Data in the Deployment Phase

The `VAEAC` method can be extended to also support incomplete data in the deployment phase, and we now sketch how this can be done. This was not discussed by Ivanov et al.

(2019), as conditioning on missing features could not occur in the original application of `VAEAC`. They used the `VAEAC` framework to impute missing feature values in a data set conditioned on the observed feature values. While in the Shapley value framework, we consider all feature combinations and can therefore end up in a setting where we need to condition on missing features.

Assume that we are in the deployment phase and consider the same setup as in Figure 6, but where the third feature is now missing, that is, $x_3 = \texttt{NA}$. The current version of `VAEAC` would essentially model $p(x_2|x_1, x_3)$ by $p(x_2|x_1)$. This naive approach is also applicable for the competing conditional methods in Section 4, but it can lead to incorrect inference when the data is not *missing completely at random* (MCAR). MCAR is a strong assumption to make about real-world incomplete data (Van Buuren, 2018). A weaker assumption is *missing at random* (MAR), that is, the missingness can be fully accounted for by the observed features. In the MAR setting, knowing that $x_3 = \texttt{NA}$ could be related to the value of x_2. Thus, when modelling x_2, we need to condition on $x_3 = \texttt{NA}$ as the missingness can contain information. For example, if x_2 is often negative when x_3 is missing, but positive when x_3 is not missing, then knowing that $x_3 = \texttt{NA}$ should influence the estimation of x_2.

We propose to extend the `VAEAC` framework by also including a missing feature mask $\mathcal{I}$ in the masked encoder, similar to what Ivanov et al. (2019) did for the full encoder, see Section 3.5. The masked encoder would then receive the observed features $\boldsymbol{x}_{\mathcal{S}} = \{x_1, 0\}$ with $x_3 = \texttt{NA}$ set to zero, the unobserved feature mask $\bar{\mathcal{S}} = \{2\}$, and the missing feature mask $\mathcal{I} = \{3\}$. To ensure missing feature masks of equal length, we would apply the same approach we used for the fixed-length unobserved feature masks $\bar{\mathcal{S}}$ above. Thus, the fixed-length input to the masked encoder would be $\{x_1, 0, 0, 0, 1, 0, 0, 0, 1\}$. Here, the first three elements correspond to the fixed-length notation of $\boldsymbol{x}_{\mathcal{S}}$, the next three corresponds to $\bar{\mathcal{S}}$, and the last three elements corresponds to $\mathcal{I}$.

Appendix D. Simulation Study: Categorical Data

Let $\{\boldsymbol{x}^{(i)}\}_{i=1,\dots,N_{\text{test}}}$ be the set of all unique M-dimensional categorical observations where each feature has L categories. Thus, $N_{\text{test}} = L^M$, which is manageable for small dimensional settings. We replace N_{test} in the denominator of EC1 (8) by $p(\boldsymbol{x}^{(i)})$, the corresponding probability mass function, and move it inside the summation. In larger dimensional settings, we use a subset of the $N_{\text{test}} = 2000$ most likely feature combinations and scale the probabilities such that they sum to 1 over those combinations.

The categorical data generating process and related Shapley value computations are elaborated in Appendix E.3. The main point there is that the categorical data is generated by sampling from a multivariate Gaussian $\mathcal{N}_M(\boldsymbol{\mu}, \Sigma_\rho)$ before categorizing each of the M features into L categories at certain cut-off-values, defined in Table 10. The covariance matrix Σ_ρ, which reflects the feature dependence, is 1 on the diagonal and ρ off-diagonal.

The categorical simulation study follows the setup of Redelmeier et al. (2020). Their results are based on a single run, which can lead to incorrect conclusions as the results vary significantly between each repetition, hence, we conduct repeated simulations. Similarly to the mixed data studies in Section 4.3, we compare the `VAEAC` approach with the `independence` and `ctree` methods.

M	L	N_{test}	Categorical cut-off values
3	3	27	$(-\infty, 0, 1, \infty)$
3	4	64	$(-\infty, -0.5, 0, 1, \infty)$
4	3	81	$(-\infty, 0, 1, \infty)$
5	6	2000	$(-\infty, -1.5, -1, 0, 0.5, 1, \infty)$
7	4	2000	$(-\infty, -0.5, 0.5, 1, \infty)$
10	3	2000	$(-\infty, -0.5, 1, \infty)$

Table 10: Outline of the setup of the categorical simulation studies in Appendix D. The fourth column describes the cut-off values between the different categories.

We consider six different setups which are described in Table 10. We use $N_{\text{train}} = 1000$ training observations to fit the predictive model, while $\rho \in \{0.0, 0.1, 0.3, 0.5, 0.8, 0.9\}$ and $\boldsymbol{\mu} = \mathbf{0}$. The response is generated according to

$$y_i = \alpha + \sum_{j=1}^{M} \sum_{l=1}^{L} \beta_{jl} \mathbf{1}(x_{ij} = l) + \epsilon_i, \tag{15}$$

where x_{ij} is the jth feature value of the ith training observation and $\epsilon_i \sim \mathcal{N}(0, 0.1^2)$, for $i = 1, \ldots, N_{\text{train}}$. The parameters α, β_{jl} are sampled from $\mathcal{N}(0, 1)$, for $j = 1, \ldots, M$ and $l = 1, \ldots, L$. The predictive model f takes the same form as (15), but without the noise term. The model parameters are estimated using standard linear regression.

The results of the categorical experiments are shown in Table 11. For all dimensions, we see that the `independence` approach is the best for independent data ($\rho = 0$). However, both `ctree` and `VAEAC` significantly outperform `independence` for moderate to strong correlation ($\rho \geq 0.3$), which is common in real-world data. For the smaller experiments ($M \leq 4$), we see that `ctree` performs better than `VAEAC` in most settings, except for $\rho = 0.5$ in two of the experiments. We have no intuitive explanation for why `VAEAC` outperforms `ctree` for this particular correlation. For the larger experiments with moderate to strong dependence, we see that `VAEAC` more often outperforms `ctree` than for the smaller experiments, especially for $M = 7$. However, for independent data, `VAEAC` is outperformed by `independence` and `ctree` in all experiments.

Appendix E. Data Generating Processes of Dependent Data

Here we present how the data are generated for the three types of simulation studies investigated in this article. In addition, we explain how we calculate the associated true Shapley values.

E.1 Continuous Data

The continuous multivariate Burr distribution is chosen as its conditional distributions have known analytical expressions and are samplable. This allows us to compute the true contribution functions and Shapley values with arbitrary precision. The Burr distribution

				$\overline{\text{EC1}}$ for each ρ					
M	L	R	Method	0.0	0.1	0.3	0.5	0.8	0.9
3	3	20	`Indep.`	**0.0170**	**0.0357**	0.0946	0.1488	0.2584	0.3623
			`Ctree`	0.0229	0.0378	**0.0428**	**0.0353**	**0.0298**	**0.0277**
			`VAEAC`	0.0395	0.0423	0.0483	0.0414	0.0425	0.0428
3	4	20	`Indep.`	**0.0177**	**0.0390**	0.1038	0.1482	0.2905	0.3733
			`Ctree`	0.0231	0.0393	**0.0529**	0.0512	**0.0485**	**0.0401**
			`VAEAC`	0.0396	0.0479	0.0533	**0.0501**	0.0508	0.0536
4	3	20	`Indep.`	**0.0182**	**0.0412**	0.1259	0.1822	0.2610	0.3590
			`Ctree`	0.0213	0.0432	**0.0488**	0.0480	**0.0417**	**0.0334**
			`VAEAC`	0.0387	0.0475	0.0514	**0.0474**	0.0446	0.0389
5	6	10	`Indep.`	**0.0263**	0.0519	0.1244	0.1908	0.3299	0.4176
			`Ctree`	0.0266	**0.0404**	**0.0636**	0.0743	**0.0841**	0.0980
			`VAEAC`	0.0620	0.0714	0.0725	**0.0720**	0.0859	**0.0748**
7	4	10	`Indep.`	**0.0189**	0.0449	0.1210	0.2020	0.3410	0.4005
			`Ctree`	0.0206	**0.0446**	0.0652	0.0706	0.0714	0.0736
			`VAEAC`	0.0513	0.0584	**0.0631**	**0.0655**	**0.0642**	**0.0647**
10	3	10	`Indep.`	**0.0163**	0.0551	0.1252	0.2260	0.4028	0.3995
			`Ctree`	0.0169	**0.0439**	0.0579	0.0618	**0.0613**	0.0578
			`VAEAC`	0.0498	0.0490	**0.0514**	**0.0566**	0.0662	**0.0561**

Table 11: Table presenting the average EC1 for the different methods applied on the categorical data simulations with correlation ρ. The bolded numbers denote the smallest average MAE per experiment and ρ, that is, the best approach.

allows for heavy-tailed and skewed marginals and nonlinear dependencies, which can be found in real-world data sets.

Other multivariate distributions with closed-form conditional distributions exist, for example, the multivariate Gaussian and the generalized hyperbolic (GH), which are fairly similar. Therefore, both give an unfair advantage to the `Gaussian` and `copula` approaches which assume Gaussian data. However, in a similar experiment to that of Section 4.2.1, not reported here, we generated data according to a GH distribution instead. The `VAEAC` approach was still able to perform on par with the `Gaussian` and `copula` methods, and much better than the `independence`, `empirical`, and `ctree` approaches.

The density of an M-dimensional Burr distribution is

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

for $x_m > 0$ (Takahasi, 1965). 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 $p(x_1, \ldots, x_S | x_{S+1} = x^*_{S+1}, \ldots, x_M = x^*_M)$, where $\boldsymbol{x}^*$ indicates the conditional values, is an S-dimensional Burr density. The associated parameters are then $\tilde{\kappa}, \tilde{b}_1, \ldots, \tilde{b}_S$, and $\tilde{r}_1, \ldots, \tilde{r}_S$, 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 $j = 1, 2, \ldots, S$. These conditional distributions are then used to compute the true contribution functions $v_{\texttt{true}}(\mathcal{S})$, for $\mathcal{S} \in \mathcal{C}$, and the Shapley values $\boldsymbol{\phi}_{\texttt{true}}$, as described in Sections 4.1 and 4.2.

An outline of one repetition of the continuous simulation study for a fixed M is given below. All sampling is done with replacements and equal probability, and are identical for the different values of N_{train} such that we can see the effect of the training size. This is repeated R times and the average evaluation criteria are presented in Tables 1 and 2.

1. Generate training and test Burr data according to Burr$(\kappa, \boldsymbol{r}, \boldsymbol{b})$. Where r_j and b_j are sampled from $\{1, 1.25, \ldots, 5\}$ and $\{2, 2.25, \ldots, 6\}$, respectively, for $j = 1, 2, \ldots, M$. Transform each feature by $u_j = F_j(x_j)$, where F_j is the true parametric (cumulative) distribution function for the jth feature x_j.

2. Generate the response y according to

$$y = \sum_{k=1}^{M/5} \left[\sin(\pi c_{k+1} u_{k+1} u_{k+2}) + c_{k+2} u_{k+3} \exp\{c_{k+3} u_{k+4} u_{k+5}\} \right], \tag{16}$$

   where c_k is sampled from $\{0.1, 0.2, \ldots, 2\}$. Add noise to (12) by sampling $M/5$ features $u^{(1)}, \ldots, u^{(M/5)}$ and creating the noise according to $\frac{\epsilon}{M/5} \sum_{k=1}^{M/5} u^{(k)}$, where $\epsilon \sim \mathcal{N}(0, 1)$.

3. Fit a random forest with 500 trees using the R-package `ranger` (Wright and Ziegler, 2017) with default parameter settings to the training data.

4. Sample $N_{\mathcal{S}} = 1000$ coalitions from $\mathcal{P}(\mathcal{M})$ to constitute $\mathcal{C}$. If $N_{\mathcal{S}} \geq 2^M$, then we set $\mathcal{C} = \mathcal{P}(\mathcal{M})$. Use the test data and $\mathcal{C}$ to compute the evaluation criteria for the different approaches as described in Section 4.1.

In Tables 1 and 2, we see that the EC1 (8) increases in line with M, which is partially linked to larger responses in (16) for higher dimensions. The Shapley values for a test observation $\boldsymbol{x}$ are distributed such that $f(\boldsymbol{x}) = \phi_0 + \phi_{\texttt{q},1} + \cdots + \phi_{\texttt{q},M}$, where ϕ_0 is the expected prediction without any features, which we have set equal to $\bar{y}_{\text{train}}$ (Aas et al., 2021a). Intuitively, ϕ_j, for $j = 1, \ldots, M$, are steps taken from $\phi_0 = \bar{y}_{\text{train}}$ to the predicted value $f(\boldsymbol{x})$. The EC1 tells us the average absolute error we make in each step compared to the true Shapley value decomposition $f(\boldsymbol{x}) = \phi_0 + \phi_{\texttt{true},1} + \cdots + \phi_{\texttt{true},M}$. Thus, the magnitude of EC1 depends on the values of M, $f(\boldsymbol{x})$, and ϕ_0. If the latter two are close, which they are in the low-dimensional settings, see Figure 8, we expect the EC1 to be reasonably low. In the high-dimensional settings, they are quite different and the $\phi_{\texttt{q},j}$ values have to cover a larger distance. Furthermore, the decomposition consist of more terms to precisely estimate. Figure 8 displays the empirical distributions of the predicted test values $f(\boldsymbol{x})$, across all repetitions, and the dashed lines represent the average ϕ_0 for the different dimensions M.

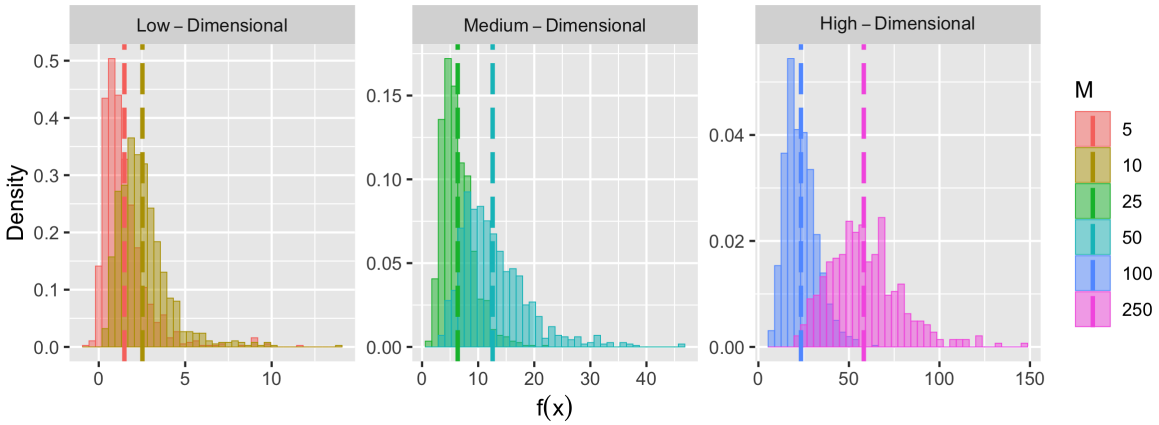

Figure 8: Empirical distributions of the predicted responses and average ϕ_0 over all repetitions.

E.2 Mixed Data

We now present a shortened description of the dependent mixed data generating process and related Shapley value computations of Redelmeier et al. (2020). For simplifying the true Shapley value computations, they use a linear predictive function of the form

$$f(\boldsymbol{x}) = \alpha + \sum_{j \in \mathcal{C}_{\text{cat}}} \sum_{l=1}^{L} \beta_{jl} \mathbf{1}(x_j = l) + \sum_{j \in \mathcal{C}_{\text{cont}}} \gamma_j x_j, \tag{17}$$

where $\mathcal{C}_{\text{cat}}$ and $\mathcal{C}_{\text{cont}}$ denote the set of categorical and continuous features, respectively, L is the number of categories for each of the categorical features, and $\mathbf{1}(x_j = l)$ is the indicator function taking the value 1 if $x_j = l$ and 0 otherwise. Furthermore, α, β_{jl} for $j \in \mathcal{C}_{\text{cat}}$, $l = 1, \ldots, L$, and γ_j for $j \in \mathcal{C}_{\text{cont}}$ are the parameters of the linear model. The dimension of the model is $M = |\mathcal{C}_{\text{cat}}| + |\mathcal{C}_{\text{cont}}|$, where $|\mathcal{C}_{\text{cat}}|$ and $|\mathcal{C}_{\text{cont}}|$ denote the number of categorical and continuous features, respectively.

To generate M-dimensional dependent mixed data, we sample from an M-dimensional Gaussian distribution $\mathcal{N}_M(\boldsymbol{\mu}, \Sigma_\rho)$, but discretize $|\mathcal{C}_{\text{cat}}|$ of the features into L categories each. Here $\boldsymbol{\mu}$ is the mean and Σ_ρ is the covariance matrix, which reflects the feature dependence, and it is 1 on the diagonal and ρ off-diagonal. That is, we first sample a random variable

$$(\tilde{x}_1, \ldots, \tilde{x}_M) \sim \mathcal{N}_M(\boldsymbol{\mu}, \Sigma_\rho),$$

and then transform the features $\tilde{x}_j$, for $j \in \mathcal{C}_{\text{cat}}$, to categorical features x_j using the following transformation:

$$x_j = l, \text{ if } v_j < \tilde{x}_j \leq v_{l+1}, \text{ for } l = 1, \ldots, L \text{ and } j \in \mathcal{C}_{\text{cat}},$$

where $v_1, \ldots, v_{L+1}$ is an increasing and ordered set of cut-off values defining the categories with $v_1 = -\infty$ and $v_{L+1} = +\infty$. We redo this N_{train} times to create the training data set of M dependent mixed features. The strength of the dependencies between the features is controlled by the correlation ρ specified in Σ_ρ. Furthermore, the actual value of x_j is irrelevant and the features are treated as non-ordered categorical features.

Computing the conditional expectation is troublesome for the mixed data setting, but the linear assumption of the predictive function in (17) simplifies the computations. As the predictive function is linear, the conditional expectation reduces to a linear combination of two types of univariate expectations. Let $\mathcal{S}_{\text{cat}}$ and $\bar{\mathcal{S}}_{\text{cat}}$ refer to the $\mathcal{S}$ and $\bar{\mathcal{S}}$ part of the categorical features $\mathcal{C}_{\text{cat}}$, respectively, with analogous sets $\mathcal{S}_{\text{cont}}$ and $\bar{\mathcal{S}}_{\text{cont}}$ for the continuous features. We can then write the desired conditional expectation as

$$\begin{aligned}
\mathbb{E}\left[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] &= \mathbb{E}\left[\alpha + \sum_{j\in\mathcal{C}_{\text{cat}}}\sum_{l=1}^{L}\beta_{jl}\mathbf{1}(x_j = l) + \sum_{j\in\mathcal{C}_{\text{cont}}}\gamma_j x_j \,\middle|\, \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \\
&= \alpha + \sum_{j\in\mathcal{C}_{\text{cat}}}\sum_{l=1}^{L}\beta_{jl}\mathbb{E}\left[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] + \sum_{j\in\mathcal{C}_{\text{cont}}}\gamma_j\mathbb{E}\left[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] \\
&= \alpha + \sum_{j\in\bar{\mathcal{S}}_{\text{cat}}}\sum_{l=1}^{L}\beta_{jl}\mathbb{E}\left[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] + \sum_{j\in\mathcal{S}_{\text{cat}}}\sum_{l=1}^{L}\beta_{jl}\mathbf{1}(x_j^* = l) \\
&\quad + \sum_{j\in\bar{\mathcal{S}}_{\text{cont}}}\gamma_j\mathbb{E}\left[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] + \sum_{j\in\mathcal{S}_{\text{cont}}}\gamma_j x_j^*.
\end{aligned}$$

To calculate the two conditional expectations, we use results from Arellano-Valle et al. (2006) on Gaussian selection distributions in addition to basic probability theory and numerical integration. Specifically, the conditional expectation for the continuous features takes the form

$$\begin{aligned}
\mathbb{E}\left[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] &= \int_{-\infty}^{\infty} x p(x_j = x|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)\,\mathrm{d}x \\
&= \int_{-\infty}^{\infty} x p(x)\frac{p(\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*|x_j = x)}{p(\boldsymbol{x}_{\mathcal{S}})}\,\mathrm{d}x,
\end{aligned} \tag{18}$$

where $p(x)$ denotes the density of the standard Gaussian distribution, $p(\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*|x_j = x)$ is the conditional distribution of $\boldsymbol{x}_{\mathcal{S}}$ given $x_j = x$, and $p(\boldsymbol{x}_{\mathcal{S}})$ is the marginal distribution of $\boldsymbol{x}_{\mathcal{S}}$. The latter two are both Gaussian and can be evaluated at the specific vector $\boldsymbol{x}_{\mathcal{S}}^*$ using the R-package `mvtnorm`, see Genz and Bretz (2009). The integral is solved using numerical integration.

For the conditional expectation of the categorical features, recall that $x_j = l$ corresponds to the original Gaussian variable $\tilde{x}_j$ falling into the interval $(v_l, v_{l+1}]$. Thus, the conditional expectation takes the form

$$\begin{aligned}
\mathbb{E}\left[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] &= P(v_l < \tilde{x}_j \leq v_{l+1}|\boldsymbol{x}_{\mathcal{S}}) \\
&= \int_{v_l}^{v_{l+1}} p(x)\frac{p(\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*|x_j = x)}{p(\boldsymbol{x}_{\mathcal{S}})}\,\mathrm{d}x,
\end{aligned}$$

which can be evaluated similarly to (18) and solved with numerical integration. Once we have computed the necessary conditional expectations for each of the 2^M feature subsets $\mathcal{S}$, we compute the Shapley values using (1). This is done for both the mixed and categorical data simulation studies.

E.3 Categorical Data

The dependent categorical data is generated similarly to the mixed data in the previous section, except that $|\mathcal{C}_{\text{cat}}| = M$. That is, we discretize all the M features into L categories.

To calculate the true Shapley values $\phi_{j,\text{true}}(\boldsymbol{x}^{[i]})$, for $j = 1, \ldots, M$ and $i = 1, \ldots, N_{\text{test}}$, we need the true conditional expectation for all feature subsets $\mathcal{S}$. When all the features are categorical, the conditional expectation can be written as

$$\mathbb{E}\left[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*\right] = \sum_{\boldsymbol{x}_{\bar{\mathcal{S}}} \in \mathcal{X}_{\bar{\mathcal{S}}}} f(\boldsymbol{x}_{\mathcal{S}}^*, \boldsymbol{x}_{\bar{\mathcal{S}}}) p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*),$$

where $\mathcal{X}_{\bar{\mathcal{S}}}$ denotes the feature space of the feature vector $\boldsymbol{x}_{\bar{\mathcal{S}}}$ which contains $L^{|\bar{\mathcal{S}}|}$ unique feature combinations.[7] We need the conditional probability $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ for each combination of $\boldsymbol{x}_{\bar{\mathcal{S}}} \in \mathcal{X}_{\bar{\mathcal{S}}}$. These can be written as $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}) = p(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})/p(\boldsymbol{x}_{\mathcal{S}})$, and then evaluated at the desired $\boldsymbol{x}_{\mathcal{S}}^*$. Since all feature combinations correspond to hyper-rectangular subspaces of Gaussian features, we can compute all joint probabilities exactly using the cut-offs $v_1, \ldots, v_{L+1}$:

$$p(x_1 = l_1, \ldots, x_M = l_M) = P(v_{l_1} < \tilde{x}_1 \leq v_{l_1+1}, \ldots, v_{l_M} < \tilde{x}_M \leq v_{l_M+1}),$$

for $l_j = 1, \ldots L$ and $j = 1, \ldots, M$. Here p denotes the joint probability mass function of $\boldsymbol{x}$ while P denotes the joint continuous distribution function of $\tilde{\boldsymbol{x}}$. The probability on the right-hand side is easily computed using the cumulative distribution function of the multivariate Gaussian distribution in the R-package `mvtnorm`. The marginal and joint probability functions based on only a subset of the features are computed analogously based on a subset of the full Gaussian distribution, which is also Gaussian.

Appendix F. Number of Monte Carlo Samples K

In Figure 9, we see how the number of Monte Carlo samples K effect the evaluation criteria in the $M = 10$-dimensional continuous simulation study, see Section 4.2.1. It is evident that $K > 250$ marginally decreases the evaluation criteria for this setting, but at the cost of higher CPU times. In practice, with limited time to generate the estimated Shapley values, we find $K = 250$ to be sufficient. We have used $K = 250$ in all experiments in this article.

[7] Redelmeier et al. (2020) incorrectly write $|\bar{\mathcal{S}}|^L$.

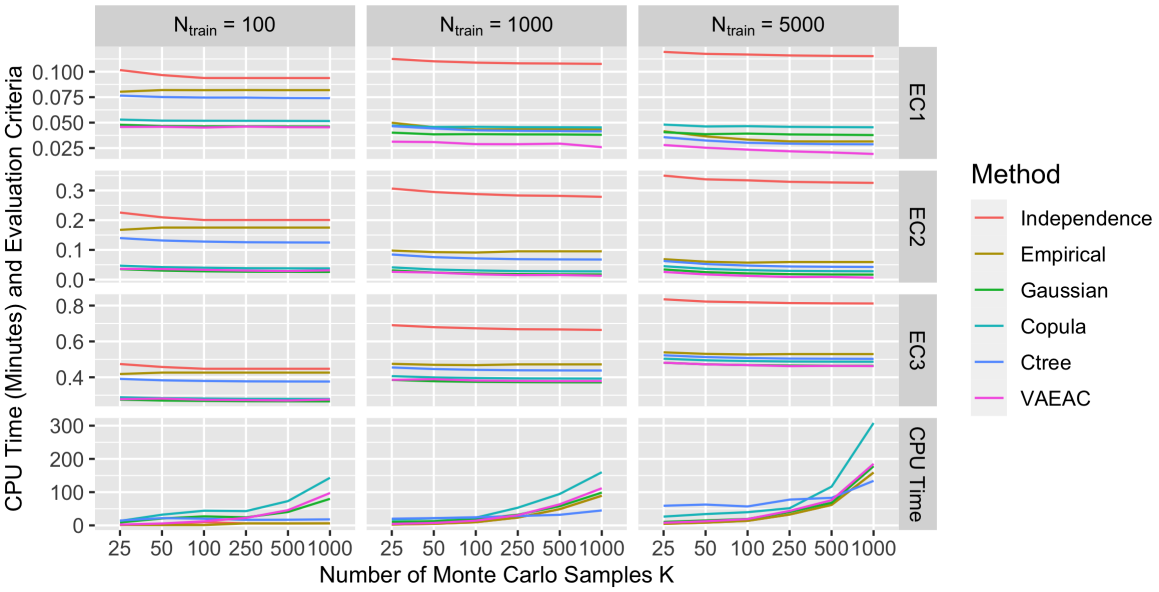

Figure 9: Same set up as in the $M = 10$-dimensional setting in Table 1, but with different numbers of Monte Carlo samples K. The CPU times are given in minutes.

Appendix G. The VAEAC Model as a Universal Approximator

Here we extend the setting of Figure 5, and illustrate the VAEAC model as a K-component Gaussian mixture model, where K is the number of Monte Carlo samples. Figure 10 illustrates, for different value of K, the K different conditional distributions $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{S}}|\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S}) = \mathcal{N}\big(\boldsymbol{x}_{\bar{S}}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S}), \text{diag}[\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S})]\big)$ inferred by the latent variables $\boldsymbol{z}^{(k)}$. The gray dots are the training data and the orange dots are the estimated means $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S})$, while the variances $\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S})$ are reflected by the contour lines. It is evident that the means follow the center of the training data, while the variance follows the spread of the training data. To better visualize the distributions simultaneously in the same figure, each $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{S}}|\boldsymbol{z}^{(k)}, \boldsymbol{x}_S, \bar{S})$ is scaled to have mode equal to 1. The contour lines for $K = 250$ are visually similar to those obtained when using 2D kernel density estimation (not illustrated here).

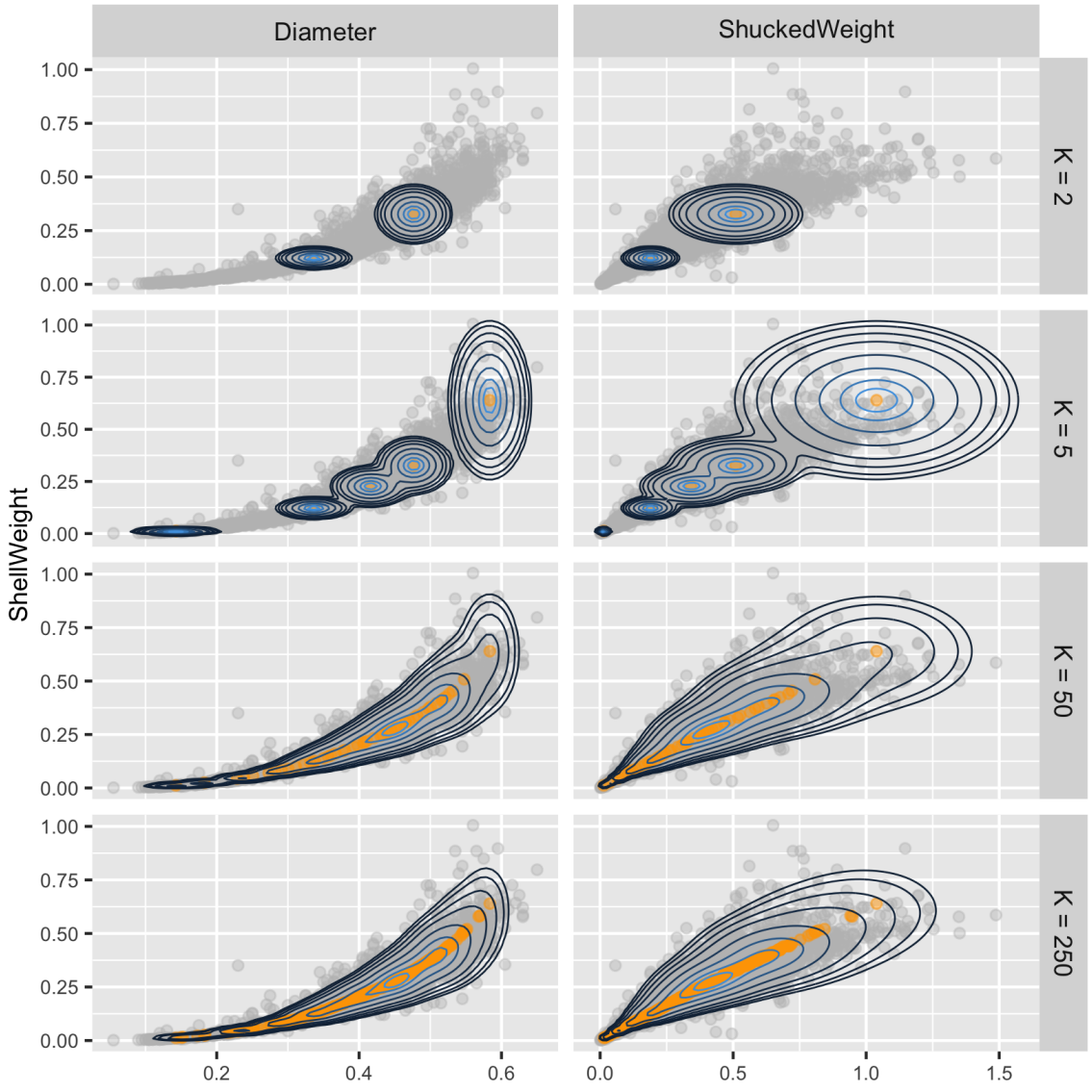

Figure 10: Extension of Figure 5 and the Abalone data set from Section 5. Illustration of the `VAEAC` model as a K-component Gaussian mixture model.

References

Kjersti Aas, Martin Jullum, and Anders Løland. Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, 298:103502, 2021a.

Kjersti Aas, Thomas Nagler, Martin Jullum, and Anders Løland. Explaining predictive models using shapley values and non-parametric vine copulas. *Dependence Modeling*, 9 (1):62–81, 2021b.

Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

Reinaldo B. Arellano-Valle, Máaarcia D. Branco, and Marc G. Genton. A unified view on skewed distributions arising from selections. *Canadian Journal of Statistics*, 34(4): 581–601, 2006.

David J Bartholomew, Martin Knott, and Irini Moustaki. *Latent variable models and factor analysis: A unified approach*. John Wiley & Sons, 2011.

Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1): 1–127, jan 2009. ISSN 1935-8237.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Herve Bourlard and Y Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59:291–4, 02 1988.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21. Association for Computational Linguistics, 2016.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3), jun 2011. ISSN 0004-5411.

Bo Chang and Harry Joe. Prediction based on conditional distributions of vine copulas. *Computational Statistics & Data Analysis*, 139:45–63, 2019.

Hugh Chen, Joseph D. Janizek, Scott Lundberg, and Su-In Lee. True to the model or true to the data? *arXiv preprint arXiv:2006.16234*, 2020.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

Ian Covert, Scott M Lundberg, and Su-In Lee. Understanding global feature contributions with additive importance measures. *Advances in Neural Information Processing Systems*, 33, 2020.

Ian Covert, Scott Lundberg, and Su-In Lee. Explaining by removing: A unified framework for model explanation. *Journal of Machine Learning Research*, 22(209):1–90, 2021.

Bin Dai, Yu Wang, John Aston, Gang Hua, and David Wipf. Connections with robust pca and the role of emergent sparsity in variational autoencoder models. *Journal of Machine Learning Research*, 19(41):1–42, 2018.

Ming Ding. The road from mle to em to vae: A brief tutorial. *AI Open*, 2021. ISSN 2666-6510.

European Commission. 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*, 2016.

Christopher Frye, Colin Rowat, and Ilya Feige. Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability. *Advances in Neural Information Processing Systems*, 33, 2020.

Christopher Frye, Damien de Mijolla, Tom Begley, Laurence Cowton, Megan Stanley, and Ilya Feige. Shapley explainability on the data manifold. In *International Conference on Learning Representations*, 2021.

Daniel Fryer, Inga Strümke, and Hien Nguyen. Shapley values for feature selection: The good, the bad, and the axioms. *IEEE Access*, 9:144352–144360, 2021. doi: 10.1109/ACCESS.2021.3119110.

Alan Genz and Frank Bretz. *Computation of multivariate normal and t probabilities*, volume 195. Springer Science & Business Media, 2009.

Paolo Giudici and Emanuela Raffinetti. Shapley-lorenz explainable artificial intelligence. *Expert Systems with Applications*, 167:114104, 2021.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Mengzhuo Guo, Qingpeng Zhang, Xiuwu Liao, and Youhua Chen. An interpretable machine learning framework for modelling human decision behavior. *arXiv preprint arXiv:1906.01233*, 2019.

Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

Sergiu Hart. Shapley value. In *Game Theory*, pages 210–216. Springer, 1989.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Tom Heskes, Evi Sijben, Ioan Gabriel Bucur, and Tom Claassen. Causal shapley values: Exploiting causal knowledge to explain individual predictions of complex models. *Advances in Neural Information Processing Systems*, 33, 2020.

Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15 (3):651–674, 2006.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational autoencoder with arbitrary conditioning. In *International Conference on Learning Representations*, 2019.

Harry Joe. Families of m-variate distributions with given margins and m (m-1)/2 bivariate dependence parameters. *Lecture Notes-Monograph Series*, pages 120–141, 1996.

Ulf Johansson, Cecilia Sönströd, Ulf Norinder, and Henrik Boström. Trade-off between accuracy and interpretability for predictive in silico modeling. *Future medicinal chemistry*, 3(6):647–663, 2011.

Martin Jullum, Anders Løland, Ragnar Bang Huseby, Geir Ånonsen, and Johannes Lorentzen. Detecting money laundering transactions with machine learning. *Journal of Money Laundering Control*, 2020.

Martin Jullum, Annabelle Redelmeier, and Kjersti Aas. Efficient and simple prediction explanations with groupshapley: A practical perspective. In *CEUR Workshop Proceedings of the XAI.it 2021 - Italian Workshop on Explainable Artificial Intelligence*, pages 1–15, 2021.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Diederik P. Kingma and M. Welling. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, 12:307–392, 2019.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Konstantina Kourou, Themis P. Exarchos, Konstantinos P. Exarchos, Michalis V. Karamouzis, and Dimitrios I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

Håvard Kvamme, Nikolai Sellereite, Kjersti Aas, and Steffen Sjursen. Predicting mortgage default using convolutional neural networks. *Expert Systems with Applications*, 102:207–217, 2018.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Yi Luo, Huan-Hsin Tseng, Sunan Cui, Lise Wei, Randall K. Ten Haken, and Issam El Naqa. Balancing accuracy and interpretability of machine learning approaches for radiation treatment outcomes modeling. *BJR— Open*, 1(1):20190021, 2019.

Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in neural information processing systems*, 29:2802–2810, 2016.

Luke Merrick and Ankur Taly. The Explanation Game: Explaining Machine Learning Models Using Shapley Values. In *Machine Learning and Knowledge Extraction*, Lecture Notes in Computer Science, pages 17–38, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57321-8.

Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. Sampling permutations for shapley value estimation, 2022.

Ghaida Riyad Mohammed, Jaffa Riad Abu Shbikah, Mohammed Majid Al-Zamili, Bassem S. Abu-Nasser, and Samy S. Abu-Naser. Predicting the age of abalone from physical measurements using artificial neural network. *International Journal of Academic and Applied Research (IJAAR)*, 4(11), 2020.

Christoph Molnar. *Interpretable Machine Learning*. lulu.com, 2019.

Mustafa Mustafa, Deborah Bard, Wahid Bhimji, Zarija Lukić, Rami Al-Rfou, and Jan M. Kratochvil. Cosmogan: creating high-fidelity weak lensing convergence maps using generative adversarial networks. *Computational Astrophysics and Cosmology*, 6(1):1, 2019.

Warwick J. Nash, Tracy L. Sellers, Simon R. Talbot, Andrew J. Cawthorn, and Wes B. Ford. The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, 48:p411, 1994.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

Art B. Owen. Sobol' indices and shapley value. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):245–251, 2014.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

Kedar Potdar, Taher S. Pardawala, and Chinmay D. Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4):7–9, 2017.

Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2020. URL `https://www.python.org/`.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org/`.

Annabelle Redelmeier, Martin Jullum, and Kjersti Aas. Explaining predictive models with mixed features using shapley values and conditional inference trees. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 117–137. Springer, 2020.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *International conference on machine learning*, pages 4364–4373. PMLR, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Alvin E. Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Egemen Sahin, Can Jozef Saul, Eran Ozsarfati, and Alper Yilmaz. Abalone life phase classification with deep learning. In *2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, pages 163–167. IEEE, 2018.

Nikolai Sellereite and Martin Jullum. shapr: An r-package for explaining machine learning models with dependence-aware shapley values. *Journal of Open Source Software*, 5(46): 2027, 2019. Version 0.2.0.

Lloyd S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2 (28):307–317, 1953.

Ian Simon, Adam Roberts, Colin Raffel, Jesse Engel, Curtis Hawthorne, and Douglas Eck. Learning a latent space of multitrack measures, 2018.

James S. Smith, Bo Wu, and Bogdan M. Wilamowski. Neural network training with levenberg–marquardt and adaptable weight compression. *IEEE transactions on neural networks and learning systems*, 30(2):580–587, 2018.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *The Journal of Machine Learning Research*, 11:1–18, 2010.

Erik Strumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

Koiti Takahasi. Note on the multivariate burr's distribution. *Annals of the Institute of Statistical Mathematics*, 17(1):257–260, 1965.

Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Kevin Ushey, J.J. Allaire, and Yuan Tang. *reticulate: Interface to 'Python'*, 2020. URL `https://CRAN.R-project.org/package=reticulate`. R package version 1.18.

Stef Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

Giulia Vilone, Lucas Rizzo, and Luca Longo. 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*, pages 85–96. Technological University Dublin, 2020.

Brian Williamson and Jean Feng. Efficient nonparametric statistical inference on population feature importance using shapley values. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10282–10291. PMLR, 13–18 Jul 2020.

Marvin N. Wright and Andreas Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

Gholamhossein Yari and Ali Mohammad Jafari. Information and covariance matrices for multivariate pareto (iv), burr, and related distributions. *International Journal of Industrial Engineering & Production Research*, 17:61–69, 2006.

Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698. PMLR, 2018.