Supplement to *A Comparative Study of Methods for Estimating Model-Agnostic Shapley Value Explanations*

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

[1]Department of Mathematics, University of Oslo, Oslo, Norway.
[2]The Alan Turing Institute, London, United Kingdom.
[3]Norwegian Computing Center, Oslo, Norway.

*Corresponding author(s). E-mail(s): lholsen@math.uio.no;
Contributing authors: glad@math.uio.no; jullum@nr.no;
kjersti@nr.no;

In Section S1, we elaborate on approaches for estimating conditional Shapley value explanations used in the main text and describe other methods. We provide additional simulation studies in Section S2. In Section S3, we include plots of some simulated and real-world data sets. We apply additional methods to the real-world experiments and decompose the computation times of the methods in Section S4. Finally, in Section S5, we provide a schematic overview of the conditional Shapley value explanation framework and the estimation methods within the explainable artificial intelligence field.

S1 Additional Approaches

In this section, we provide more information about the methods used in the main text, describe additional approaches we have used, and point out potential methods that can be incorporated into the Shapley value explanation framework in the future.

S1.1 The Missingness During Training Procedure

Covert et al. (2021, Appendix E.2) and Chen et al. (2022a, Section. 5.1.3) describe a procedure where they directly estimate the conditional expectation by modifying the training process of the predictive model f such that it handles missing features. That is, they train f on a particular objective function such that its predictions of observations with missing features are equivalent to marginalizing out the features using the conditional distribution. However, as we focus on model-agnostic post hoc explanation for arbitrary f and most predictive models do not support missingness, we skipped this procedure in the main part of the article.

S1.2 The Generative Method Class

Here, we describe alternative versions of the `VAEAC` approach that we have considered and then list other potential methods in the `generative` method class.

S1.2.1 VAEAC with Response Feature

The `VAEAC-f` approach includes the predicted response of the predictive model $\hat{y} = f(\boldsymbol{x})$ as an additional feature that is always unobserved in the deployment phase. That is, the extended training data takes the form $\{\boldsymbol{x}^{[i]}, f(\boldsymbol{x}^{[i]})\}_{i=1}^{N_{\text{train}}}$. This idea was proposed by Ivanov et al. (2019), the creators of the `VAEAC` methodology, and they argued that this extension could improve the modeling of the data, especially for multi-modal data. The `VAEAC-f` approach will generate $(\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}, \hat{y}_{\mathcal{S}}^{(k)}) \sim p_{\boldsymbol{\psi},\boldsymbol{\theta}}(\boldsymbol{x}_{\bar{\mathcal{S}}}, y_{\mathcal{S}} | \boldsymbol{x}_{\mathcal{S}}, \bar{\mathcal{S}})$, for which two possible procedures are available. First, in the indirect `VAEAC-f-indir` approach, we only use the $\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}$ part, combine them with $\boldsymbol{x}_{\mathcal{S}}$, send them through the predictive model f, and finally estimate the contribution function with $\hat{v}(\mathcal{S}, \boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K} f(\boldsymbol{x}_{\bar{\mathcal{S}}}^{(k)}, \boldsymbol{x}_{\mathcal{S}})$. For the other approach, which we call the direct `VAEAC-f-dir` approach, we skip the intermediate step where we evaluate the model at the Monte Carlo samples by rather using the $\hat{y}_{\mathcal{S}}^{(k)}$ samples, that is, $\hat{v}(\mathcal{S}, \boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K} \hat{y}_{\mathcal{S}}^{(k)}$. This saves time if f is computationally expensive to call. We use the same hyperparameters for these two approaches as for the original `VAEAC` approach and the estimated model parameters at the epoch with the lowest validation error; see Appendix A. We consider several maximum numbers of epochs and indicate this by including the number in the method name. For example, `VAEAC-f-dir-500` means that we trained the `VAEAC-f-dir` method for 500 epochs.

S1.2.2 VAEAC with Paired Sampling

The `VAEAC-paired` approach is identical to the `VAEAC` method described in Section 3.4.2, except that we used paired sampling when generating the mask. That means that both $\mathcal{S}$ and $\bar{\mathcal{S}}$ are applied to the same observation in the training phase of the `VAEAC` model.

S1.2.3 Potential Generative Methods

We can use various applicable generative methods to generate the conditional Monte Carlo samples and Shapley values, such as non-parametric vine copulas (Aas et al., 2021). We now provide a non-exhaustive list of other applicable generative methods, which, to the best of our knowledge, have yet to be used in Shapley value estimation. Computing the Monte Carlo samples coincides with the field of *multiple imputation of missing values*. The methods in this rich field can be categorized into two classes (Zheng and Charoenphakdee, 2022). The first class contains the iterative approaches: the Multivariate Imputations based on Chained Equations (`MICE`) (Van Buuren and Groothuis-Oudshoorn, 2011) and `MissForest` (Stekhoven and Bühlmann, 2011). The second class contains the deep generative models: Multiple Imputation using Denoising Autoencoders (`MIDA`) (Gondara and Wang, 2018), Missing Data Importance-weighted Autoencoder (`MIWAE`) (Mattei and Frellsen, 2019), Generative Adversarial Imputation Nets (`GAIN`) (Yoon et al., 2018), and Conditional Score-based Diffusion Models for Tabular data (`CSDI_T`) (Zheng and Charoenphakdee, 2022). Other methods are the Arbitrary Conditioning Flow model (`ACFlow`) (Li et al., 2020), the Neural Conditioner (`NC`) (Belghazi et al., 2019), Neural Autoregressive Distribution Estimation (`NADE`) (Uria et al., 2016), and Universal Marginalizers (`UM`) (Douglas et al., 2017).

S1.3 The Separate and Surrogate Method Class

In this section, we describe additional regression-based approaches. Note that all the regression methods can, in theory, be used both in the `separate` and `surrogate regression` frameworks. Some of them might, however, be infeasible for the latter in practice due to memory or time constraints, especially for large training data sets and high dimensions. Note that not all the regression methods minimize the mean squared error loss function.

S1.3.1 Polynomial Regression

Polynomial regression is an extension of linear regression where we model the relationship as a pth degree polynomial for each feature. That is, the model takes the following form:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \left(\beta_{j,1} x_j^1 + \beta_{j,2} x_j^2 + \cdots + \beta_{j,p} x_j^p \right) = \beta_0 + \sum_{j=1}^{M} \sum_{k=1}^{p} \beta_{j,k} x_j^k.$$

We estimate the coefficients of the polynomial model using the `lm` function in base R with `formula = "y ~ poly(X1, deg = p) +...+ poly(XM, deg = p)"`, where `p` is the degree. We call the approach `Poly-p`.

S1.3.2 Linear Regression with Interactions

We extend the linear regression model by including interactions between the features. For example, we get the following model formula when we include first-order interactions:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \beta_j x_j + \sum_{j=1}^{M-1} \sum_{k=j+1}^{M} \beta_{j,k} x_j x_k.$$

For second-order interactions, we get the following model:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \beta_j x_j + \sum_{j=1}^{M-1} \sum_{k=j+1}^{M} \beta_{j,k} x_j x_k + \sum_{j=1}^{M-2} \sum_{k=j+1}^{M-1} \sum_{l=k+1}^{M} \beta_{j,k,l} x_j x_k x_l.$$

We estimate the coefficients in the interaction model using the `lm` function in base R with `formula = "y ~ (.)^o+1"`, where `o` is the order. We call the approach `LM-inter-o`.

S1.3.3 Polynomial Regression with Interactions

Here, we extend the linear regression model with interactions by also allowing for polynomial terms. For example, the polynomial regression model with polynomial degree 2 and interactions of one order lower takes the following form:

$$f(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{M} \beta_j x_j + \sum_{j=1}^{M} \sum_{k=j}^{M} \beta_{j,k} x_j x_k.$$

We estimate the coefficients in the polynomial interaction model using the `lm` function in base R with `formula = "y ~ poly(X1, ..., XM, deg = d)"`, where `d` is the degree. For the surrogate version, we do not include interactions with the binary mask features, as the number of coefficients to be estimated drastically increases. We call the approach `Poly-inter-d`.

S1.3.4 Generalized Additive Models

We also fit GAMs using the `gam` package (Hastie, 2022), which differs slightly from the `mgcv` package discussed in Appendix A. In the `gam` package, we can directly specify the degrees of freedom for the splines. We consider three different versions: one with `df` $= 5$ (`GAM-5`), another with `df` $= 10$ (`GAM-10`), and in the last, we conduct cross-validation using the `caret` package to tune the degrees of freedom (`GAM-CV`).

S1.3.5 Elastic Net Regression

Elastic Net models add regularization to the model coefficients, and the popular Lasso and Ridge regression models are special cases. However,

they do not minimize the MSE, but we still include them. The objective function for the Gaussian family is: $\min_{(\beta_0,\boldsymbol{\beta})\in\mathbb{R}^{p+1}} \frac{1}{2N}\sum_{i=1}^{N}(y_i-\beta_0-\boldsymbol{x}_i^T\boldsymbol{\beta})^2+\lambda\left((1-\alpha)\|\boldsymbol{\beta}\|_2^2/2+\alpha\|\boldsymbol{\beta}\|_1\right)$, where $\lambda \geq 0$ is a regularization parameter and $0 \leq \alpha \leq 1$ is a compromise between Ridge ($\alpha = 0$) and Lasso regression ($\alpha = 1$). We consider $\alpha \in \{0, 0.5, 1\}$ and call the corresponding methods for `Ridge`, `Elastic`, and `Lasso`. We use the `glmnet` package (Friedman et al., 2010) to fit the models and use the package's cross-validation procedure to tune λ.

S1.3.6 Principal Component Regression

The difference between regular linear regression and principal component regression (PCR) is that the latter regress the response on the principal components instead of the original features. For more details, see Hastie et al. (2009, pp. 79-80). We use the `pls` package (Liland et al., 2021) to fit the PCR model and use the package's cross-validation procedure to determine the number of principal components to include in the final model. We call the approach `PCR`.

S1.3.7 Partial Least Squares

The partial least squares (PLS) regression model is similar to PCR, but the PLS also uses the response when constructing the linear combinations of the features for regression. For more details, see Hastie et al. (2009, pp. 80-82). We use the `pls` package (Liland et al., 2021) to fit the PLS model and use the package's cross-validation procedure to determine the number of components to include in the final regression model. We call the approach `PLS`.

S1.3.8 Projection Pursuit Regression

Section 3.5.3 and Appendix A describe the PPR model and explain that we use cross-validation to determine the number of terms L in the `PPR separate` approach. An alternative is the `PPR-fixed separate` approach where we let $L = |\mathcal{S}|$. This method is much faster than `PPR separate` but still competitive; see Section S2. For larger values of M, letting $L = \min(|\mathcal{S}|, L_{\max})$, where $L_{\max}$ is a threshold, will reduce the computation time even more.

S1.3.9 Support Vector Machines

Support vector machines (SVM) used for regression are also known as support vector regression, and see Hastie et al. (2009, Ch. 12.3) for an introduction. We use ϵ-type regression and a radial kernel, but one could also consider ν-regression and linear, polynomial, and sigmoid kernels. We use the `WeightSVM` package (Xu et al., 2021) to fit the SVM, as it supports weighting of the observations, but the more well-known `e1071` package (Meyer et al., 2022) could also have been used. We call the approach `SVM`.

S1.3.10 K-Nearest Neighbors

The K-nearest neighbors (KNN) regression model is a memory-based approach that does not require any model to be fit. For a given individual $\boldsymbol{x}^*$, the model finds the K closes observations in the training data and return the mean response of these observations. We use the `kknn` package (Schliep and Hechenbichler, 2016) to train the KNN model and to conduct hyperparameter tuning. We call the approach `KNN`.

S1.3.11 Single Decision Tree

A regression decision tree partitions the feature space into a set of rectangles and predicts a new observation's response as the mean response of the training observations in the particular partition. CART, C4.5, and CTree are popular methods for tree-based regression, which are very simple to understand yet powerful; see, e.g., Hastie et al. (2009, Ch. 9.2). We use the `rpart` package (Therneau and Atkinson, 2022) to fit a decision tree with complexity parameter 0.001 and then prune the tree afterward. We call the approach `Tree`.

S1.3.12 Random Forest

In the main text, the `RF` approach was tuned using cross-validation, which leads to a large computation time. Here we propose a default approach called `RF-def` with 500 trees and default hyperparameter values in the `ranger` package (Wright and Ziegler, 2017). A potential improvement is to vary the number of trees based on the coalition size $|\mathcal{S}|$ instead of having a fixed number as, e.g., 500 trees might be excessive when $\mathcal{S}$ is a singleton.

S1.3.13 Boosting

Both `XGBoost` (Chen et al., 2015) and `CatBoost` (Prokhorenkova et al., 2018) are gradient-based boosted decision trees, but they differ in that the latter supports categorical data by default while the former require the user to do, e.g., one-hot encoding. We include two versions of the `XGBoost` approach: one where we use default hyperparameter values (`XGBoost-def`) and one where we tune `nrounds`, `max_depth`, `eta`, `gamma`, `colsample_bytree` using the default grid in the `caret` package. We call the latter approach `XGBoost`, which is time-consuming due to the extensive hyperparameter tuning.

S1.3.14 Neural Networks and Multilayer Perceptron

In Sections S2 and S4, we explore `NN surrogate` methods with hyperparameters defined in Section 5 but with different maximum numbers of learning epochs. E.g., `NN-Olsen-500 surrogate` indicates that `num_epochs` = 500, but we still use the network weights at the epoch with the lowest validation error. We have also implemented a version that employs early stopping, as discussed in Appendix A. More precisely, we stop the training

if no improvement has been made to the validation error in 150 epochs. Additionally, this version initiates ten networks to reduce the likelihood of poorly initiated network parameters, as discussed in Section S4. We train the ten networks for fifteen epochs and continue only with the network with the lowest validation error. We denote this method by `NN-Olsen-ES` and `NN-Frye-ES`.

The multilayer perceptron (MLP) is a fully connected feedforward artificial neural network. We include some small networks to compare these against the large `NN-Olsen surrogate` and `NN-Frye surrogate` methods described in Section 3.6.2 and Appendix A. We call the approach, e.g., `NN-[`u`, `v`, `w`]`, which means that the network has three layers where the number of neurons in the layers are u, v, and w, respectively. We use the `RSNNS` package (Bergmeir and Benítez, 2012), with default hyperparameters and 200 epochs.

S1.3.15 Potential Methods

Bénard et al. (2022) use a projected random forest to estimate Shapley effects, which is not the same as Shapley values in (1). The projected random forest is a `surrogate regression` model that provides predictions of the output conditioned on any feature subset. Thus, their procedure can be adapted to estimate conditional Shapley values. Jethani et al. (2021) propose another neural network-based procedure that skips the modeling of the data/response altogether by training a complex neural network that takes in the full input feature vector $\boldsymbol{x}$ and directly outputs the Shapley values $\boldsymbol{\phi}$.

S1.3.16 Surrogate Regression Methods in High-Dimensions

In high-dimensional settings, to reduce the computational cost, one can consider training the `surrogate regression` model on a sampled subset of the augmented representations in (5). In that case, one should ensure that all coalitions are present. Uniform sampling will mostly sample coalitions with approximately half of the features present, as the number of coalitions with $|\mathcal{S}|$ entries is given by $\binom{M}{|\mathcal{S}|}$. Therefore, Covert et al. (2021) propose to first uniformly sample the coalition size, i.e., $|\mathcal{S}| \sim \mathcal{U}[1, M-1]$, and then sample $|\mathcal{S}|$ features with uniform probability. Recall that the number of terms in the Shapley value formula in (1) grows at an exponential rate; hence, in higher dimensions, it is common practice to estimate the Shapley values based on a sampled collection of coalitions with replacement (Chen et al., 2022a; Lundberg and Lee, 2017; Olsen et al., 2022). We can then create $\mathcal{X}_{\text{aug}}$ only based on these coalitions. Furthermore, many regression models support weighting of the observations, which we can set as the sampling frequency of the different coalitions. Olsen et al. (2022) use a similar idea when sampling masks. For regression models that do not support weights, one can duplicate the relevant data, but this naïve approach increases the number of rows in the augmented design matrix.

S2 Additional Simulation Studies

In this section, we extend the numerical simulation studies in Section 4 in two directions. First, in Section S2.1, we include more setups with Gaussian data. Second, in Section S2.2, we use the multivariate Burr distribution instead of the multivariate Gaussian.

S2.1 Gaussian Distributed Experiments

Here we include additional setups to the experiments in Sections 4.1 and 4.2. We include:

`lm_some_interactions`: $f_{\text{lm,some}}(\boldsymbol{x}) = f_{\text{lm,no}}(\boldsymbol{x}) + \gamma_1 x_1 x_2$,
`lm_many_interactions`: $f_{\text{lm,many}}(\boldsymbol{x}) = f_{lm,\text{more}}(\boldsymbol{x}) + \gamma_3 x_5 x_6$,
`gam_one`: $f_{\text{gam,one}}(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{1} \beta_i \cos(x_i) + \sum_{j=2}^{M} \beta_j x_j$,
`gam_two`: $f_{\text{gam,two}}(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{2} \beta_i \cos(x_i) + \sum_{j=3}^{M} \beta_j x_j$,
`gam_five`: $f_{\text{gam,five}}(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{5} \beta_i \cos(x_i) + \sum_{j=6}^{M} \beta_j x_j$,
`gam_some_interactions`: $f_{\text{gam,some}}(\boldsymbol{x}) = f_{\text{gam,no}}(\boldsymbol{x}) + \gamma_1 g(x_1, x_2)$,
`gam_many_interactions`: $f_{\text{gam,many}}(\boldsymbol{x}) = f_{\text{gam,more}}(\boldsymbol{x}) + \gamma_3 g(x_5, x_6)$,

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

The results for the first two linear setups are very similar to those obtained in Section 4.1. The correct `parametric` approaches are the most accurate, while the `generative` method class is generally the second-best class for moderate ρ. However, the `separate regression` method class is the second-best class for higher values of ρ.

The results of the `gam_one` and `gam_two` experiments are almost identical to those obtained in the `lm_no_interactions` experiment, which is unsurprising as the setups are very similar. When we include one nonlinear term, the `LM separate` is the most accurate, but as expected, this changes when we include more nonlinear terms. In this case, the correct `parametric` methods are the most accurate, together with the `GAM separate` method. Again, we see a tendency for the `separate regression` methods to become more precise for higher dependence levels. The same holds for the `parametric` methods. The results of the `gam_five` experiment are nearly identical to those described for the `gam_three` experiment in the main text.

For the `gam_some_interactions` and `gam_many_interactions` experiments, we obtain indistinguishable results. Furthermore, the results also coincide with the results we observed in Section 4.2; see Figure S1. Generally, the `parametric` methods are superior, with the `generative` methods as a close runner-up, except for $\rho = 0.9$, where the `separate regression` methods constitute the second-best method class, in particular, the `PPR separate` method. Furthermore, using the `PPR-fixed separate` method introduced in Section S1.3.8 drastically decreases the computational cost without sacrificing much precision.

S2.2 Burr Distributed Experiments

In this section, we repeat the same simulation studies as in Section 4 and Section S2.1, but we replace the multivariate Gaussian data with multivariate Burr data. The Burr distribution is strictly positive, heavy-tailed, skewed, and has nonlinear dependence and known conditional distributions; see Appendix B.2.

We sample $N_{\text{train}} = 1000$ training and $N_{\text{test}} = 250$ test observations from a Burr$(\kappa, \boldsymbol{b}, \boldsymbol{r})$ distribution. We let $\boldsymbol{b} = \{5, 4, 6, 5, 3, 6, 5, 5\}$, $\boldsymbol{r} = \{4, 3, 5, 2, 5, 3, 5, 1\}$, while we vary the scale parameter $\kappa \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$. Here, a low κ indicates high dependency. The average Pearson correlations for the six values of κ are 0.80, 0.63, 0.46, 0.36, 0.29, and 0.25, respectively. In Figures S6 and S7 in Section S3, we display plots of the Burr data when $\kappa = 1$ and $\kappa = 3$, respectively. A larger value of κ makes the Burr distribution more Gaussian-like, while lower values of κ produce more extreme observations due to the right heavy-tailed property of the Burr distribution. We observe that the methods struggle with these extreme observations, and the individual MAE is (often) higher for these observations in the outer region of the data distribution. That is, the data has few similar observations for the methods to learn the conditional structure. In Figures S2 to S4, we present a selection of the results. The results for the other settings are very similar.

We observe similar results for the Burr data as we did for the Gaussian data. That is, using the correct `parametric` approach, in this case, the `Burr` approach, yields the most accurate Shapley value explanations. The `GH` method also performs well, even though it makes an incorrect parametric assumption, while the `Gaussian` and `copula` methods perform relatively worse. The best method outside the `parametric` method class is generally a `VAEAC` approach, where the number in the name indicates the maximum number of epochs. However, for the less complex setups without interaction terms, the `VAEAC` approaches are often outperformed by some of the `separate regression` methods, particularly the `GAM separate` and `PPR separate` approaches. We do not see a systematic benefit of choosing a large value for the maximum number of epochs in the `VAEAC` approaches. Thus, the differences are likely based on better initialized random weights in the networks, similar to what we discuss for the `NN-Olsen surrogate` method in Section S4.1. The `VAEAC-f-dir` approaches are consistently outperformed by the `VAEAC` and `VAEAC-f-indir` methods, and there does not seem to be a systematic winner between the latter two methods. Some of the additional regression-based methods proposed in Section S1.3 perform relatively well, such as the proposed `LM-inter separate` and `Poly-inter separate` methods.

For the `surrogate regression` approaches, we notice that the `LM-inter surrogate` and `Poly-inter surrogate` often outperform the complex `NN-Frye surrogate` and `NN-Olsen surrogate` approaches. However, for the complex `gam_more_interactions` experiment, the `NN-Olsen surrogate` approach with a high number of epochs is the overall best `surrogate`

Method	Time	Method	Time	Method	Time
Independence	36:23.4	Poly-inter-3 sep.	3.1	Poly-inter-2 sur.	7.0
Empirical	13:17.6	GAM sep.	43.7	Poly-inter-3 sur.	17.0
Gaussian	35:59.0	GAM-5 sep.	26.6	GAM sur.	22.8
Copula	39:14.1	GAM-10 sep.	1:37.7	GAM-5 sur.	12.3
GH	42:38.5	GAM-CV sep.	25:43.0	GAM-10 sur.	55.0
Burr	38:30.5	PCR sep.	3.9	PCR sur.	14.9
Ctree	18:45.6	PLS sep.	3.3	PLS sur.	14.1
VAEAC-200	40:20.1	PPR sep.	1:55.4	PPR sur.	4:29.9
VAEAC-500	41:58.2	PPR-fixed sep.	4.6	KNN sur.	38:10.4
VAEAC-1000	44:03.8	SVM sep.	14.0	Tree sur.	13.0
VAEAC-paired-200	52:54.4	KNN sep.	18.8	RF sur.	55:44.6
VAEAC-f-indir-200	40:53.8	Tree sep.	3.5	RF-def sur.	2:19.1
VAEAC-f-indir-500	42:04.1	RF sep.	56:04.1	XGBoost sur.	2:02:39.5
VAEAC-f-indir-1000	44:13.9	RF-def sep.	40.0	XGBoost-def sur.	50.7
VAEAC-f-dir-200	5:05.3	XGBoost sep.	26:32.7	CatBoost sur.	34.6
VAEAC-f-dir-500	6:46.0	XGBoost-def sep.	33.5	NN-[3, 4] sur.	3:48.7
VAEAC-f-dir-1000	8:54.0	CatBoost sep.	4:47.9	NN-[32, 16, 8] sur.	7:44.2
LM sep.	0.6	NN-[3, 4] sep.	1:32.1	NN-Frye sur.	13:44:46.7
LM-inter-2 sep.	0.7	NN-[32, 16, 8] sep.	5:13.1	NN-Frye-15000 sur.	14:53:22.9
LM-inter-3 sep.	0.8	LM sur.	2.7	NN-Frye-ES sur.	44:08.3
Lasso sep.	15.5	LM-inter-2 sur.	8.4	NN-Olsen sur.	7:15:32.6
Ridge sep.	17.3	Lasso sur.	5.2	NN-Olsen-2500 sur.	3:58:45.2
Elastic sep.	15.9	Ridge sur.	6.5	NN-Olsen-10000 sur.	15:22:43.1
Poly-2 sep.	0.4	Elastic sur.	4.4	NN-Olsen-ES sur.	26:40.1
Poly-3 sep.	1.7	Poly-2 sur.	4.4		
Poly-inter-2 sep.	1.9	Poly-3 sur.	5.1		

Table S1: The CPU times used by the methods to compute Shapley values for the $N_{\text{test}} = 250$ test observations in the `gam_more_interactions` experiment with Burr distributed features when $\kappa = 2$ and $N_{\text{train}} = 1000$ (Section S2.2). The format of the CPU times is hours:minutes:seconds, where we omit the larger units of time if they are zero, and the colors indicate the different method classes.

`regression` approach. For the complex setups in Section 4.2 and Section S2.1 with interactions, we also let the predictive model f be a random forest and a projection pursuit regression model, as done in Section 4.5. This had a minor effect on the overall results. However, some of the non-smooth regression methods, such as `CatBoost separate`, performed relatively better when f was also non-smooth, just as in the main text. We also looked at $N_{\text{train}} \in \{100, 5000\}$, and the overall tendencies remained.

In Table S1, we report the CPU times for the approaches used in the `gam_more_interactions` experiment with Burr distributed features, $\kappa = 2$, $N_{\text{train}} = 1000$, and $N_{\text{test}} = 250$. We see similar time tendencies as discussed in Section 4.3, but note the time differences between the `separate regression` methods with default and cross-validated versions. For example, the `RF separate` takes approximately 84 times longer than the `RF-def separate` method due to the extra cost of tuning the hyperparameters. However, the `RF separate` method obtains lower MAE scores; see Figure S4. We observe similar tendencies for the `PPR separate` and `PPR-fixed separate` methods. Here, we only report the total time, but the time decomposition is similar to the one in Table 1. That is, the `separate regression` and `surrogate regression` methods spend the vast majority of their time on training, while the predicting step only takes a couple of seconds. In contrast, the Monte Carlo-based

approaches spend most of the time on the predicting and generating steps in that order.

S3 Characteristics of the Data Sets

In this section, we provide pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients (for the continuous features) between the features in some of the data sets we have used in this article. In Figures S5 to S7, we include figures for a few of the simulated Gaussian and Burr distributed data sets from the numerical simulation studies in Section 4 and Section S2, respectively. Further, plots for the Abalone, Diabetes, and Wine data sets from Section 5 are provided in Figures S8 to S10, respectively. We omit the Adult dataset as it is chaotic and difficult to interpret due to the large number of levels in the categorical features. However, the pairwise Pearson correlation coefficients for the continuous features are all close to zero.

In Figures S5 to S7, we include plots for the Gaussian distribution with $\rho = 0.9$ and Burr distribution with $\kappa \in 1, 3$, respectively. For the Gaussian features, the pairwise correlation between features i and j is given by $\rho^{|i-j|}$, i.e., it decreases from 0.9 to 0.48 in the $M = 8$ situation. For the Burr distribution, the marginals become more right-skewed, and the correlation approximately doubles when we decrease κ from 3 to 1. The average correlations are 0.28 and 0.61 for $\kappa = 3$ and $\kappa = 1$, respectively.

For the Abalone data set (Figure S8), there is a clear nonlinearity and heteroscedasticity among the pairs of features and a significant pairwise correlation between the features. All continuous features have a pairwise correlation above 0.775, or 0.531 when grouped by the categorical feature `Sex`. There is a clear distinction between infants and females/males. All marginals are right-skewed, except for the features `Length` and `Diameter`, which is left-skewed.

The Diabetes data set (Figure S9) shows a fairly strong correlation between many features. For example, the correlation between `S1` and `S2` is 0.90. On average, the `Age` feature is the least correlated feature with the other features. Most scatter plots and marginal distributions display structures and marginals somewhat similar to Gaussian distribution, except the `S4` feature, which has a multi-modal marginal.

In contrast, for the (Red) Wine dataset (Figure S10), most scatter plots and marginal density functions display structures and marginals far from the Gaussian distribution, as most marginals are right-skewed. The largest correlation in absolute value is 0.683 (between `pH` and `fix_acid`), while most other pairs of features have zero to moderate correlation.

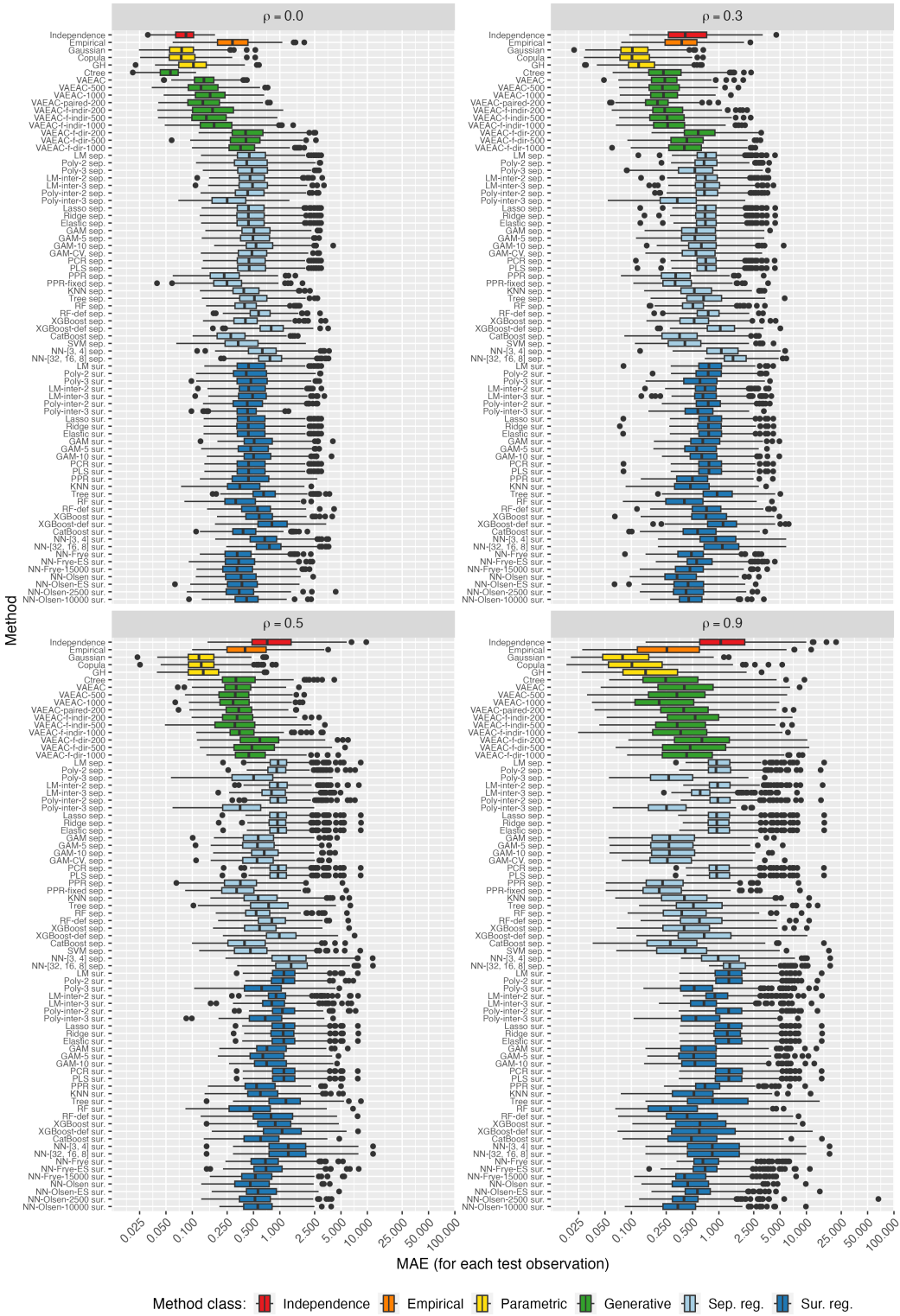

Fig. S1: Experiment `gam_many_interactions` with Gaussian data.

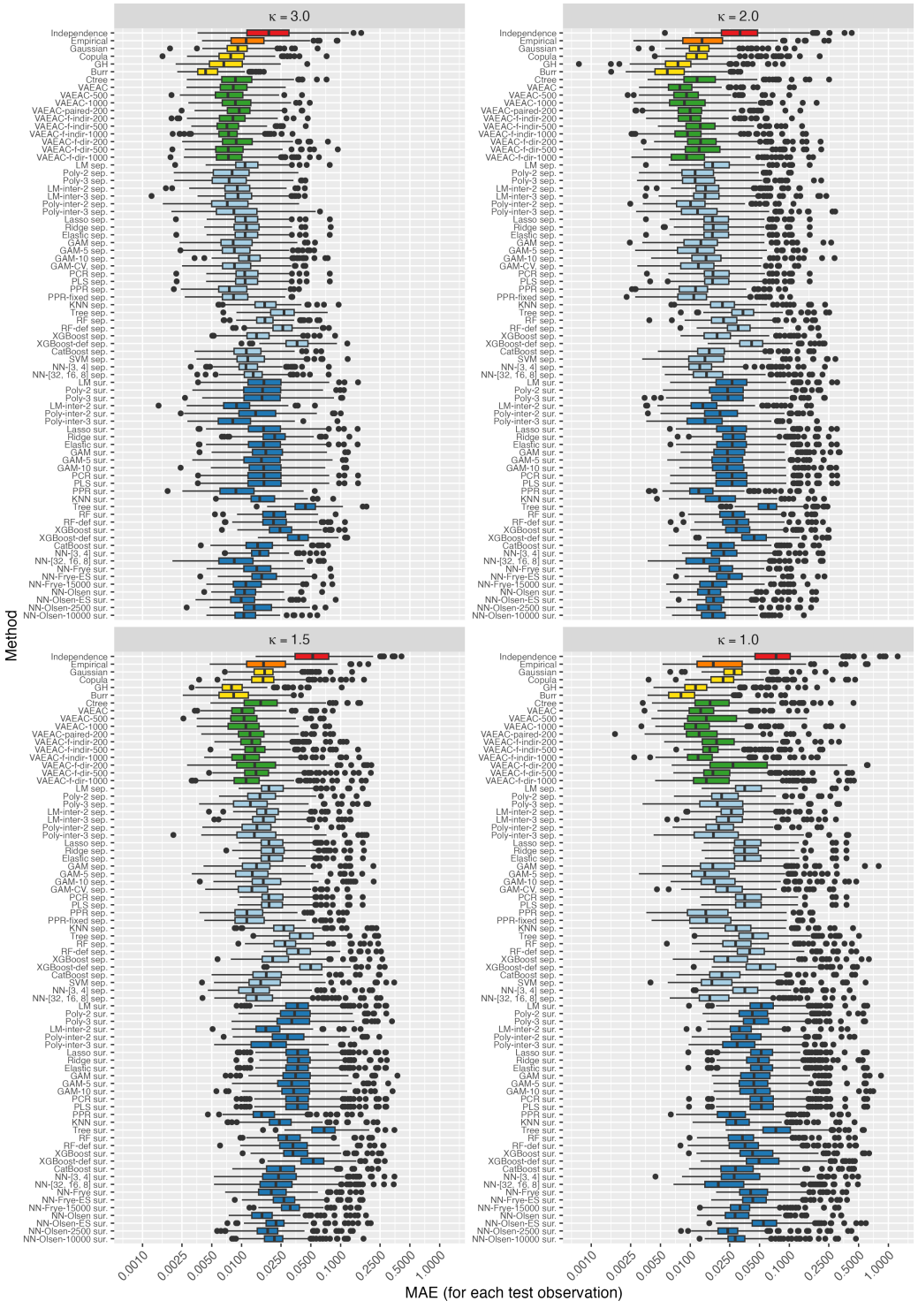

Fig. S2: Experiment `lm_more_interactions` with Burr data.

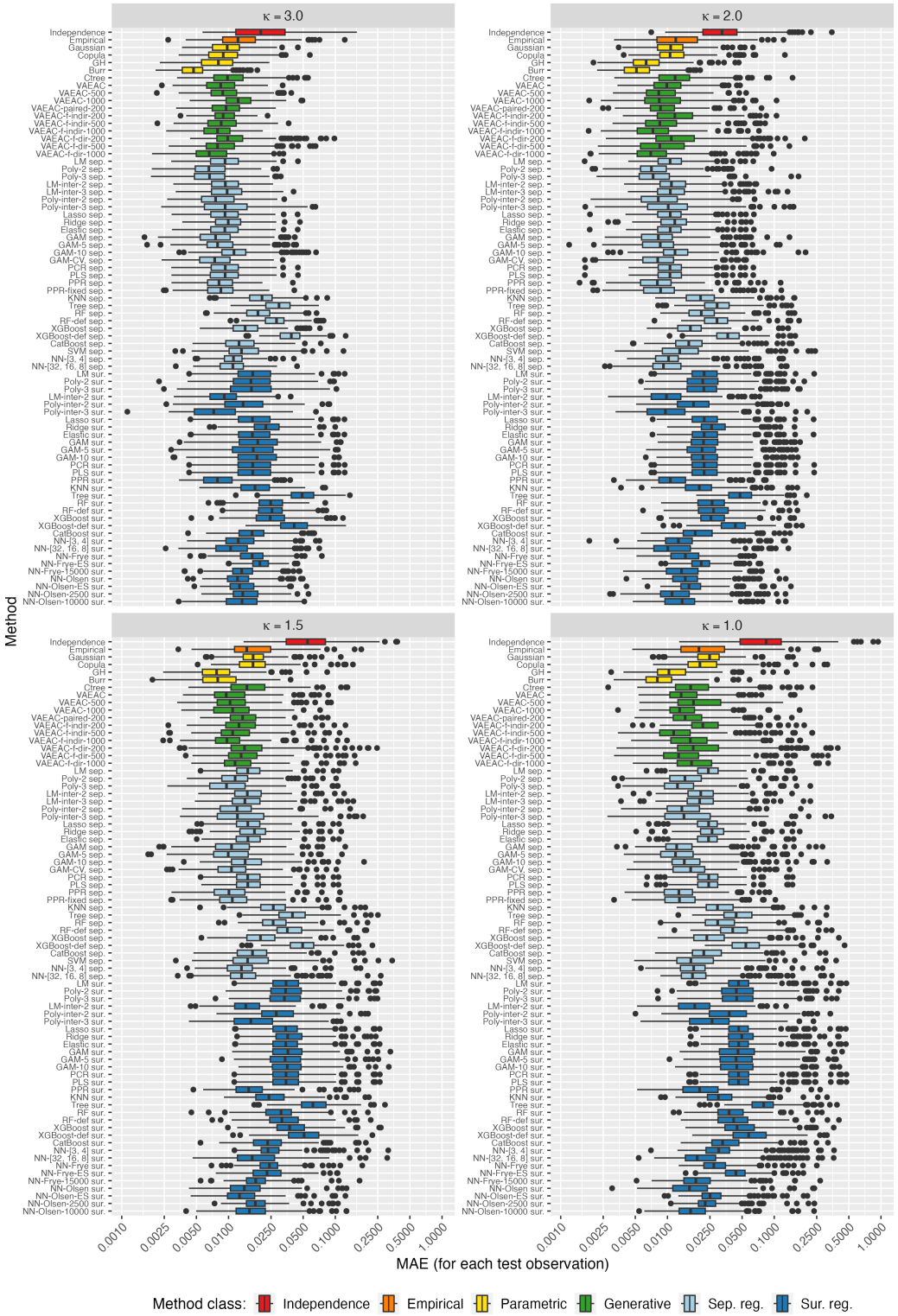

Fig. S3: Experiment `gam_three` with Burr data.

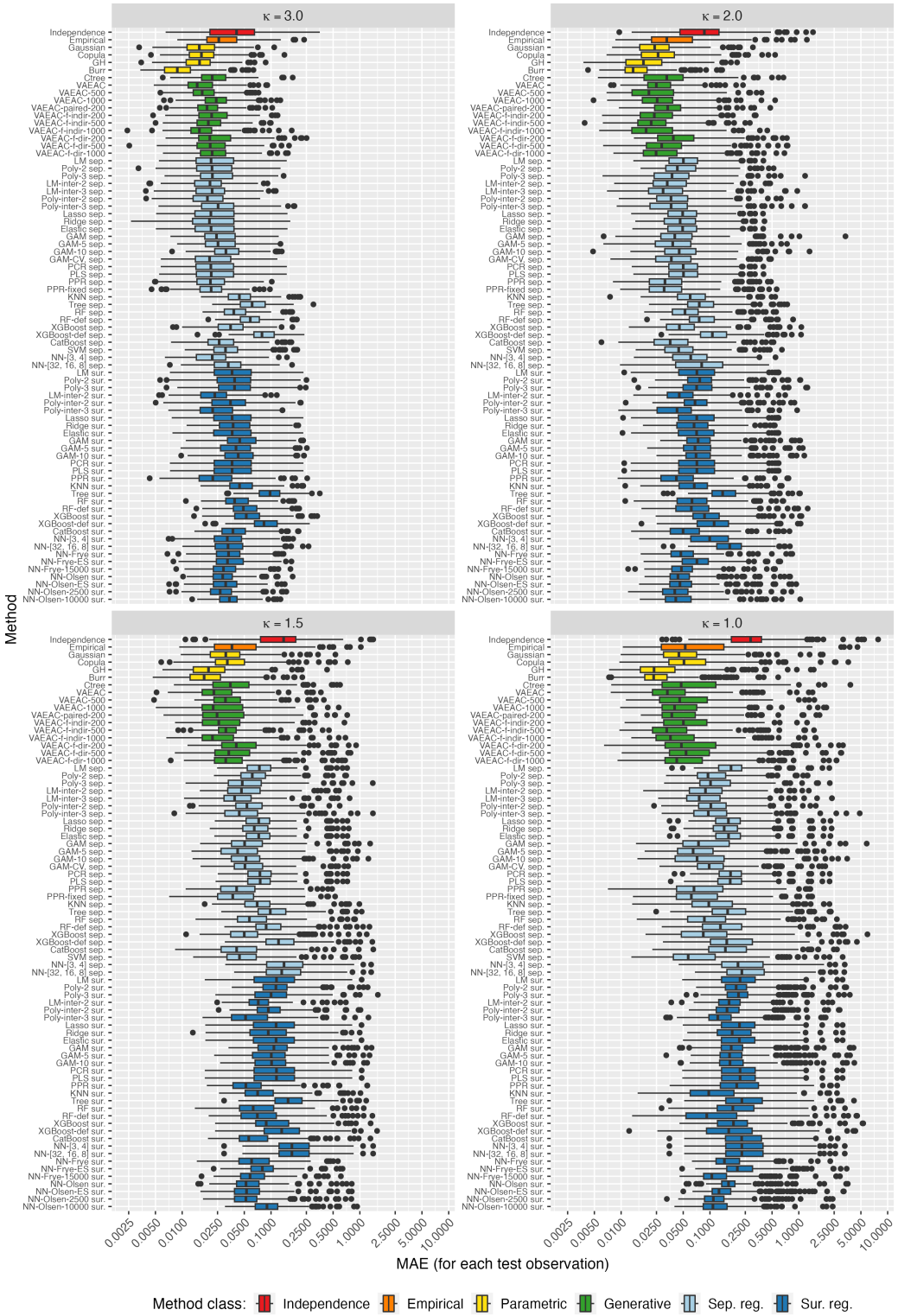

Fig. S4: Experiment `gam_more_interactions` with Burr data.

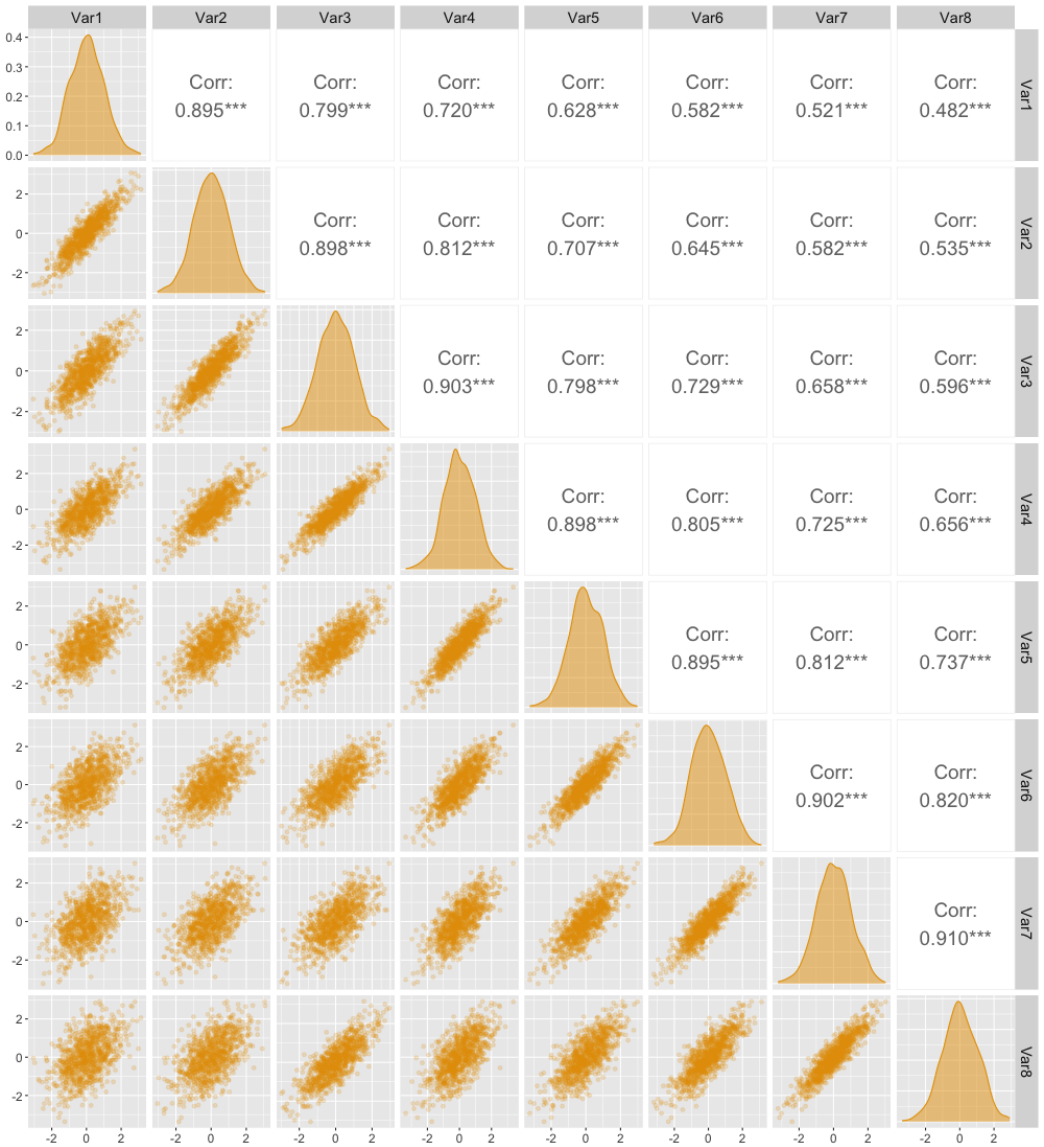

Fig. S5: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the **simulated Gaussian distributed features** in Section 4 with $\rho = 0.9$. The correlation between feature i and j is given by $\rho^{|i-j|}$.

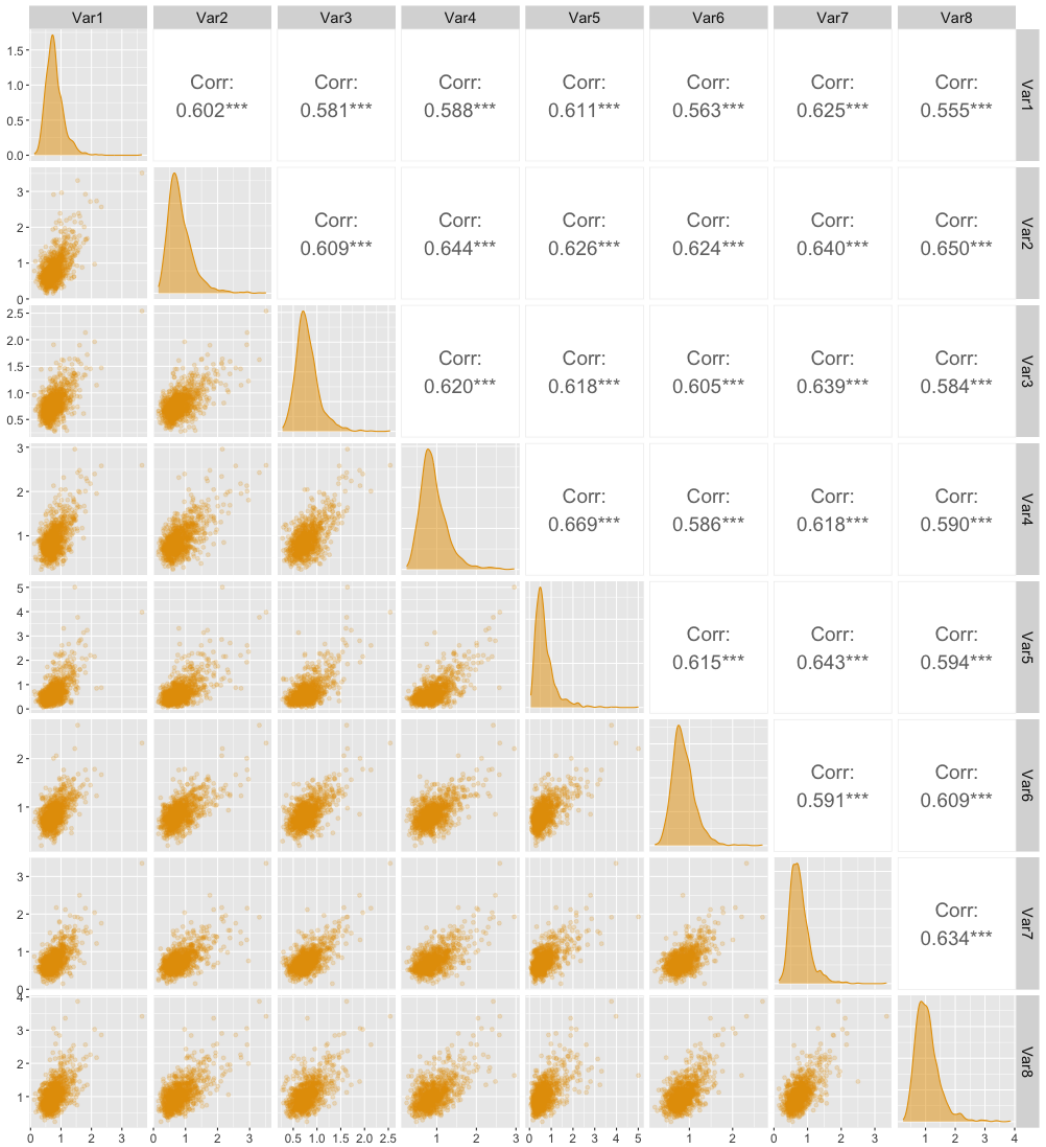

Fig. S6: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the **simulated Burr distributed features** in Section S2 with $\kappa = 1$.

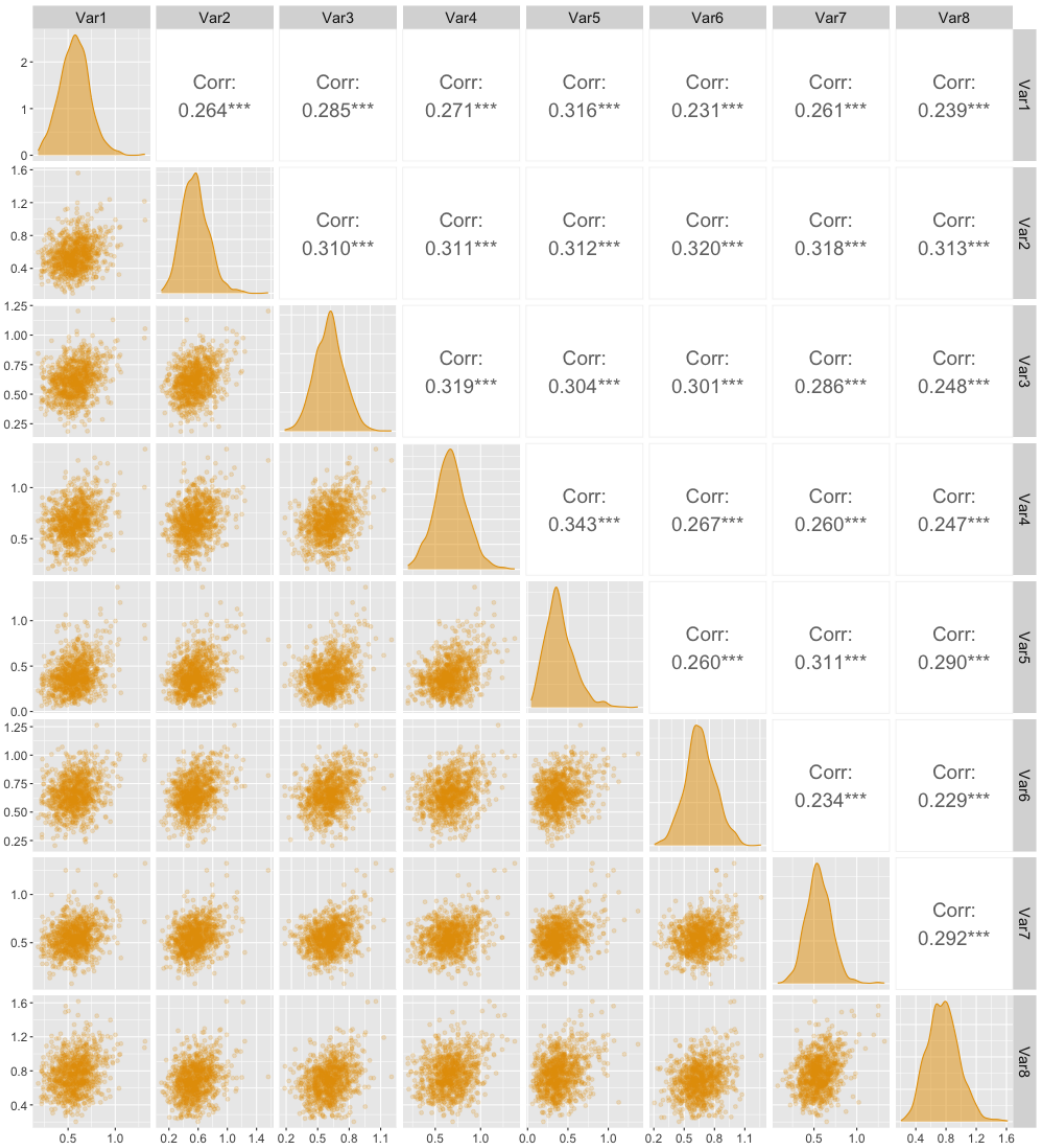

Fig. S7: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the **simulated Burr distributed features** in Section S2 with $\kappa = 3$.

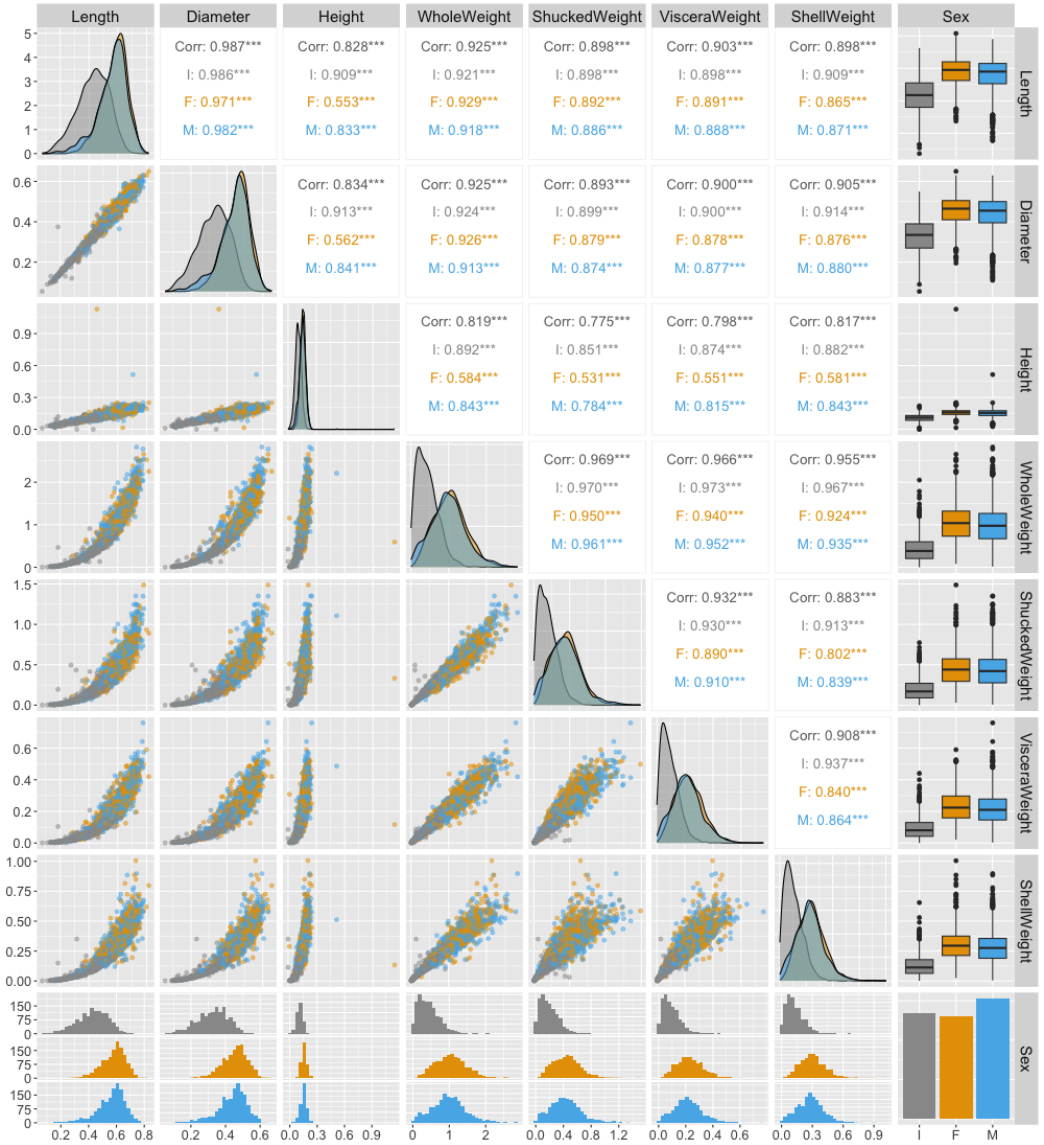

Fig. S8: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the features in the **Abalone data set** used in Section 5. The figure is grouped by `Sex`, where the infants are gray, females are yellow, and males are blue. The correlations reported in black correspond to all observations, while the colored correlations are grouped based on `Sex`.

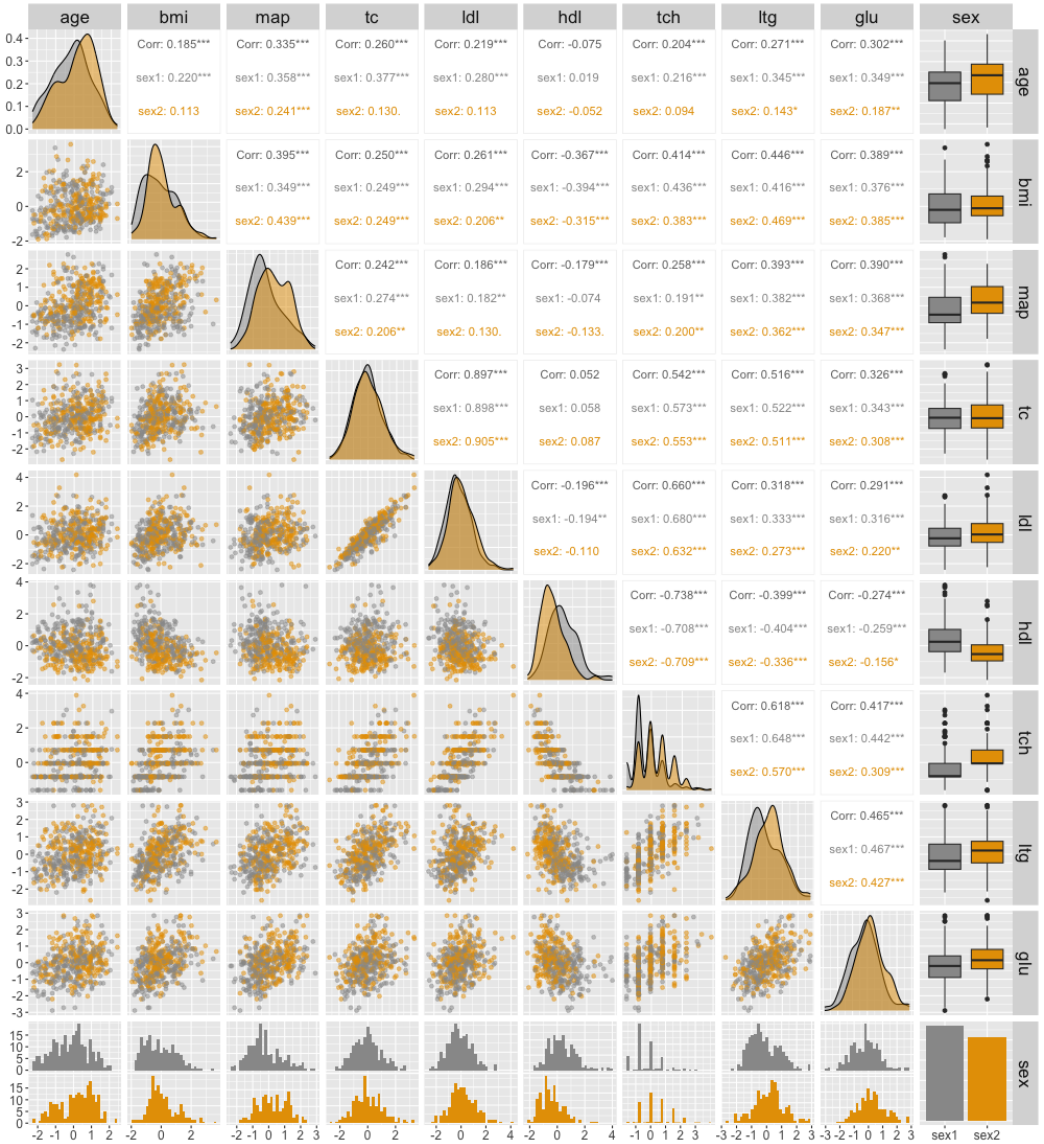

Fig. S9: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the features in the **Diabetes data set** used in Section 5. The figure is grouped by `Sex`. The correlations reported in black correspond to all observations, while the colored correlations are grouped based on `Sex`.

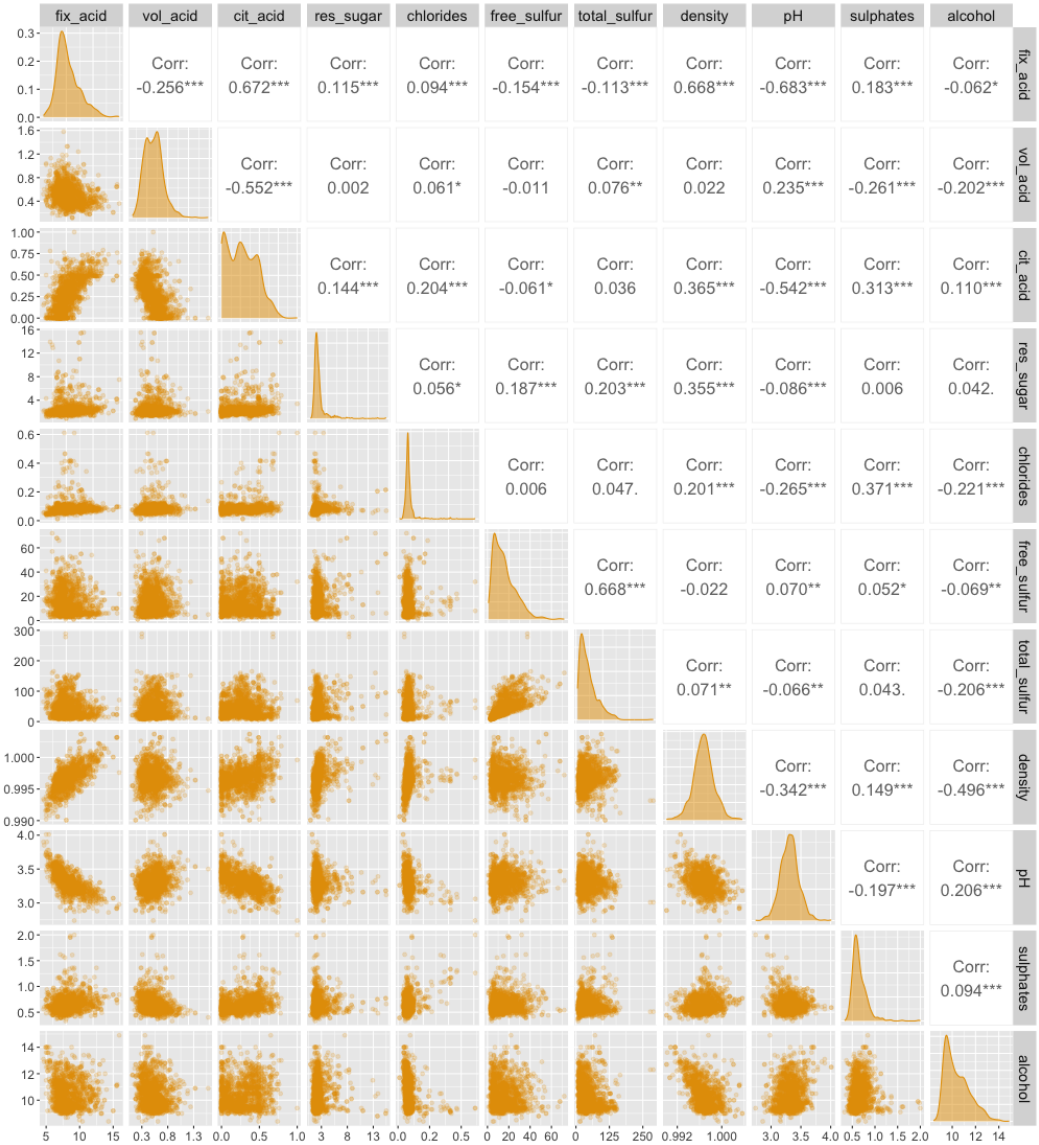

Fig. S10: Pairwise scatter plots, marginal density functions, and pairwise Pearson correlation coefficients for the features in the **(Red) Wine data set** used in Section 5.

S4 Real-World Data Experiments: Computation Time

Tables S2 to S6 present the decomposed CPU times for the Abalone, Diabetes, Wine, and Adult experiments from Section 5, respectively. The tables also include the MSE_v scores for the different methods. The total CPU times are decomposed into the same three categories as in Section 4.3: *training*, *generating*, and *predicting*. Recall that we ran the three first experiments on the MAC system specified in Section 4.3, while the Adult experiment was run on a shared computer server described in Section 5. The CPU times will differ from computer to computer.

Here, we include some methods in addition to those used in Section 5, such as the `PCR separate` approach (Section S1.3.6), which performs similarly to the best methods for the Diabetes experiment. Recall that the predictive model f in the Diabetes experiment is a PCR model. Hence, this supports our findings in the main text that we should use a `separate regression` method with the same form as f for accurate Shapley value estimates. Additionally, we include the `RF-def separate` method (Section S1.3.12) with default hyperparameters to illustrate the need for conducting cross-validation to tune the hyperparameters. We also include versions of the `VAEAC`, `NN-Olsen surrogate`, and `NN-Frye surrogate` approaches with default hyperparameters but with different numbers of training epochs. It should also be noted that the `PPR-fixed separate` method with a fixed number of terms $L = |\mathcal{S}|$ produces almost as good results as the CV-alternative `PPR separate` in a fraction of the time.

As in Section 4.3, the training step is the most time-consuming step for the `separate regression` and `surrogate regression` methods, while the predicting step often takes only a couple of seconds. For the Abalone and Diabetes datasets, creating the augmented training data set takes approximately 2 seconds, while it takes approximately 14 seconds for the Wine data set. The time increase is due to larger a M (compared to Abalone) and N_{train} (compared to Diabetes). In comparison, the Monte Carlo-based methods use most of their time generating the Monte Carlo samples in the Abalone and Diabetes experiments. This contrasts with the timings in Table 1, where the predicting step was the slowest. The time difference is caused by the GAM model in the `gam_more_interactions` experiment being more computationally expensive to call than the PPR and PCR models in Abalone and Diabetes experiments, respectively. For the Wine experiment, the predicting step is the most expensive, which is caused by the RF model being more computationally costly to call, and we have more calls due to a larger M (compared to Abalone) and N_{test} (compared to Diabetes).

When excluding the training time, which is only done once and can be considered an upfront time cost, it is evident that the regression-based methods are superior with respect to computation time. For example, consider the best Monte Carlo and regression-based methods for the $\text{Abalone}_{\text{cont}}$ experiment, i.e., the `VAEAC-10000` and `PPR separate` methods, respectively. The `VAEAC-10000` approach uses 551.9 seconds to explain 1044 predictions,

an average of 0.53 seconds per explanation. In contrast, the `PPR separate` method explains all the 1044 predictions in 0.5 seconds. Thus, there is a speed difference of a factor of 1104, which is essential when the number of predictions to explain is large.

If training time is not a limiting factor, we can use more time to train the `NN-Olsen surrogate` and `NN-Frye surrogate` methods, as these methods are slow to train but fast in the predicting step. In contrast to the numerical simulation studies in Section 4, the validation errors for some of the complex real-world experiments were still decreasing for these approaches, indicating that more training would be beneficial. Table S2 shows that the MSE_v scores for the different versions of the `NN-Olsen surrogate` approach decrease when we increase the number of epochs leading it to share the first place with the `PPR separate` approach. However, recall that we use the network at the epoch with the lowest validation error, which was the 6457th epoch for the `NN-Olsen-20000 surrogate` approach. This means that the increased performance was not due to the additional number of training epochs but rather to better random initialization values, which caused the network parameters to converge to a better local optimum. The same tendency also holds for the other real-world experiments. For example, there is essentially no difference in the MSE_v scores of the `NN-Olsen-500 surrogate` and `NN-Olsen-10000 surrogate` methods for the Diabetes data set in Table S4. Furthermore, the validation data is randomly extracted and removed from the training data for each `NN surrogate` method. Thus, it might be that for some `NN surrogate` methods and data sets, we were unlucky in that the training and validation data were not representative of the test data. This is more likely to happen for small data sets.

Table S4 shows that the `RF-def separate` approach with default hyperparameters provides almost as low MSE_v score as the cross-validated counter-version `RF separate`, whose training time is approximately 140 times longer. For the `surrogate regression` version, we see that the `RF-def surrogate` method outperforms the `RF surrogate` even though the default hyperparameters are an option in the cross-validation procedure.

S4.1 Analysis of the `NN-Olsen surrogate` method for the Abalone Data Set

In this section, we look closer at the effect of initialization values and hyperparameters for the `NN-Olsen surrogate` method for the two experiments on the Abalone data set.

For the $\text{Abalone}_{\text{all}}$ experiment, we fitted ten versions of the `NN-Olsen-10000 surrogate` method with different idealizations seeds. These networks were trained on the shared computer server described in Section 5 and had an average training time of 14:02:46.6. In comparison, the training CPU time for the `NN-Olsen-10000 surrogate` method in Table S3 was almost 3.5 times higher when trained on the MAC operating system described in Section 4.3. The average MSE_v was 1.214, with a standard deviation of

0.0085. Furthermore, on average, the best epoch was the 5509th, with a standard deviation of 1660. The large standard deviation means there is a large variability when the networks reach their minimum validation error. Seven versions reached their minimum before epoch 5000, while the slowest reached its minimum at the 8920th epoch. This means that if `num_epochs` = 5000, the performance of seven of the networks would not be influenced by the reduced number of epochs. In contrast, the precision of the three remaining versions would decrease. This highlights the need for good network initialization values when `num_epochs` is limited or for choosing a large value for `num_epochs`.

For the `NN-Olsen-10000 surrogate` method, we wanted to investigate the effect of the hyperparameters. We considered the same hyperparameter grid as in Appendix A, i.e., `lr` $\in \{0.01, 0.001, 0.0001\}$ and `width` $\in \{32, 64, 128\}$ and we call the corresponding method for `NN-Olsen-CV-10000 surrogate`. We fit ten versions of the `NN-Olsen-CV-10000 surrogate` method with different initialized network weights. The average best epoch was the 7706th, with a standard deviation of 2496, while the average MSE_v was 1.208, with a standard deviation of 0.0236. This score is nearly identical to the average score we obtained for the `NN-Olsen-10000 surrogate` method above with default hyperparameters: `lr` = 0.001 and `width` = 64. The default hyperparameters were chosen two times, but `lr` = 0.0001 and `width` = 128 was the best combination five times. For these five repetitions, we obtained an average MSE_v of 1.198, with a standard deviation of 0.0053. However, the average best epoch was then 9456, with a standard deviation of 473, which is close to the maximum number of epochs. Thus, we might see further improvements for this hyperparameter combination by increasing `num_epochs`. We ran one network with `num_epochs` = 20 000, which obtained its best validation score after 15 518 epochs. The method's training time was 1:08:07:48.1, and it got an MSE_v score of 1.214, which is at the same level as previous versions.

We repeated the investigations for the $\text{Abalone}_{\text{cont}}$ experiment. For the ten versions of the `NN-Olsen-10000 surrogate` approach, we obtain an average MSE_v score of 1.178, with a standard deviation of 0.0068. This score is lower than the one reported in Tables 2 and S2. Thus, it is likely that that version had poorly initialized network parameters or that the training and validation data sets were not representative. The average best epoch was the 5410th, with a standard deviation of 1842, and the average training time was 10:45:13.1. The best of the ten versions obtained an MSE_v score of 1.167, beating all other methods. We also fitted ten versions of the `NN-Olsen-CV-10000 surrogate` method. They obtained an average MSE_v score of 1.77, with a standard deviation of 0.0067. That is, there is minimal improvement in conducting cross-validation. The default hyperparameters were never the best hyperparameter combination. The `lr` = 0.0001 and `width` = 128 combination was the best five times, while `lr` = 0.001 and `width` = 128 was best four times. For the former combination, the average MSE_v score is 1.171, with a standard deviation of 0.0015. The average best number of epochs is 9253; meaning that we should consider increasing `num_epochs`. We ran one network with

`num_epochs` = 20 000, which obtained its best validation score after 11 523 epochs. The method's training time was 1:02:02:24.5, and it got an MSE_v score of 1.169, equal to the two best methods; `NN-Olsen-20000 surrogate` and `PPR separate`. However, the latter is approximately 700 times faster to train.

Several methods exist to stabilize and robustify neural networks: we can regularize the network parameters, apply drop-out during training, or create an ensemble model of several networks. One can also initialize several networks and only continue to train the best-performing one after a fixed number of epochs. The latter is done in the `NN-Frye-ES` and `NN-Olsen-ES surrogate` methods, but we do not see a systematic improvement. In the R package `torch` (Falbel and Luraschi, 2022), the weights and biases in each layer are uniformly initialized from $\mathcal{U}(-\sqrt{N_{\text{in}}}, \sqrt{N_{\text{in}}})$, where N_{in} is the number of inputs to the linear layers in the network. Other initialization schemes exist, such as Xavier initialization (Glorot and Bengio, 2010) and Kaiming initialization (He et al., 2015), where the latter considers the rectifier nonlinearities to initialize the network parameters robustly. Discovering better procedures and ensuring representative training, validation, and test data should be of focus and is mentioned as further work in the conclusion in Section 7.

Method	Training	Generating $\boldsymbol{x}_{\mathcal{S}}^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
`Independence`	0.0	7.6	1:16.6	1:24.2	8.679
`Empirical`	35.3	2:10.3	57.6	3:43.2	1.540
`Gaussian`	0.0	2:21.1	1:22.9	3:44.0	1.349
`Copula`	0.0	13:01.0	2:04.5	15:05.5	1.223
`GH`	4:20.2	2:53.9	1:25.6	8:39.7	1.292
`Burr`	2:56.3	1:00.1	1:25.9	5:22.3	5.640
`Ctree`	9.6	7:19.4	11.8	7:40.8	1.393
`VAEAC-200`	3:03.2	6:52.1	1:30.8	11:26.1	1.340
`VAEAC-1000`	14:52.5	7:22.2	1:25.0	23:39.7	1.217
`VAEAC-10000`	2:24:51.7	7:43.1	1:28.8	2:34:03.6	1.182
`VAEAC-20000`	4:56:25.7	7:49.5	1:20.9	5:05:36.1	1.195
`VAEAC-40000`	10:25:31.1	7:48.4	1:31.2	10:34:50.7	1.193
`VAEAC-f-indir-200`	3:27.7	9:42.7	1:27.7	14:38.1	1.412
`VAEAC-f-indir-1000`	16:23.3	7:59.2	1:27.4	25:49.9	1.255
`VAEAC-f-indir-10000`	2:36:23.3	8:23.1	1:25.8	2:46:12.2	1.197
`VAEAC-f-indir-20000`	5:10:17.5	8:17.9	1:19.0	5:19:54.4	1.184
`VAEAC-f-indir-40000`	10:32:42.9	9:08.1	1:30.6	10:43:21.6	1.181
`VAEAC-f-dir-200`	3:27.7	9:42.7	0.0	13:10.4	1.686
`VAEAC-f-dir-1000`	16:23.3	7:59.2	0.0	24:22.5	1.310
`VAEAC-f-dir-10000`	2:36:23.3	8:23.1	0.0	2:44:46.4	1.228
`VAEAC-f-dir-20000`	5:10:17.5	8:17.9	0.0	5:18:35.4	1.231
`VAEAC-f-dir-40000`	10:32:42.9	9:08.1	0.1	10:41:51.1	1.196
`LM sep.`	0.2	—	0.1	0.3	1.684
`Poly-2 sep.`	1.5	—	0.2	1.7	1.350
`Poly-3 sep.`	1.4	—	0.2	1.6	1.320
`LM-inter-2 sep.`	0.3	—	0.1	0.4	1.423
`LM-inter-3 sep.`	0.4	—	0.1	0.5	1.389
`Poly-inter-2 sep.`	1.3	—	0.4	1.7	1.320
`Poly-inter-3 sep.`	2.4	—	0.4	2.8	1.394
`Lasso sep.`	8.1	—	0.2	8.3	1.696
`Ridge sep.`	10.2	—	0.1	10.3	2.027
`Elastic sep.`	9.2	—	0.1	9.3	1.706
`GAM sep.`	27.4	—	6.3	33.7	1.298
`GAM-5 sep.`	11.4	—	1.0	12.4	1.306
`GAM-10 sep.`	19.9	—	1.0	20.9	1.294
`GAM-CV, sep.`	5:55.3	—	1.1	5:56.4	1.303
`PCR sep.`	4.8	—	0.1	4.9	1.719
`PLS sep.`	3.2	—	0.1	3.3	1.717
`PCR sep.`	4.6	—	0.1	4.7	1.721
`PPR sep.`	2:14.6	—	0.5	2:15.1	**1.169**
`PPR-fixed sep.`	7.1	—	0.4	7.5	1.270
`SVM sep.`	47.5	—	3.7	51.2	1.260
`KNN sep.`	30.2	—	4.0	34.2	1.330
`Tree sep.`	3.4	—	0.2	3.6	1.553
`RF sep.`	1:09:06.1	—	9.7	1:09:15.8	1.239
`RF-def sep.`	39.7	—	4.5	44.2	1.312
`CatBoost sep.`	6:16.9	—	0.2	6:17.1	1.190
`LM sur.`	3.2	—	0.5	3.7	2.912
`Poly-2 sur.`	4.3	—	0.8	5.1	2.664
`Poly-3 sur.`	4.1	—	0.9	5.0	2.628
`LM-inter-2 sur.`	7.2	—	0.9	8.1	1.775
`Poly-inter-2 sur.`	5.9	—	1.2	7.1	2.447
`Poly-inter-3 sur.`	15.2	—	1.7	16.9	1.930
`Lasso sur.`	7.2	—	0.2	7.4	2.912
`Ridge sur.`	7.4	—	0.1	7.5	2.998
`Elastic sur.`	7.4	—	0.2	7.6	2.912
`GAM sur.`	28.5	—	12.6	41.1	2.611
`GAM-5 sur.`	18.3	—	1.0	19.3	2.612
`GAM-10 sur.`	1:01.4	—	0.7	1:02.1	2.605
`PCR sur.`	17.8	—	0.6	18.4	2.912
`PLS sur.`	17.2	—	0.7	17.9	2.912
`PPR sur.`	14:56.2	—	1.3	14:57.5	1.548
`KNN sur.`	0.2	—	0.0	0.2	13.081
`Tree sur.`	11.2	—	0.6	11.8	2.839
`RF sur.`	1:14:15.8	—	15.0	1:14:30.8	1.281
`RF-def sur.`	1:45.1	—	6.0	1:51.1	1.448
`XGBoost sur.`	2:46:14.9	—	0.6	2:46:15.5	1.536
`XGBoost-def sur.`	1:05.5	—	0.6	1:06.1	1.437
`CatBoost sur.`	9:09.2	—	1.6	9:10.8	1.298
`NN-Frye-3000 sur.`	5:47:58.1	—	1.9	5:48:00.0	1.625
`NN-Frye-6000 sur.`	11:34:15.4	—	1.7	11:34:17.1	1.433
`NN-Frye-15000 sur.`	1:03:05:38.7	—	2.3	1:03:05:41.0	1.310
`NN-Frye-40000 sur.`	3:01:48:38.7	—	2.1	3:01:48:40.8	1.244
`NN-Frye-ES sur.`	6:10:13.2	—	2.1	6:10:15.3	1.374
`NN-Olsen-500 sur.`	2:24:15.7	—	1.7	2:24:17.4	1.248
`NN-Olsen-2500 sur.`	12:10:18.6	—	1.4	12:10:20.0	1.201
`NN-Olsen-10000 sur.`	2:00:40:20.3	—	2.0	2:00:40:22.3	1.194
`NN-Olsen-20000 sur.`	3:22:23:46.0	—	2.0	3:22:23:48.0	**1.169**
`NN-Olsen-ES sur.`	8:28:19.1	—	1.8	8:28:20.9	1.191

Table S2: $\text{Abalone}_{\text{cont}}$ data set experiment: $M = 7$, $N_{\text{train}} = 3133$, and $N_{\text{test}} = 1044$.

Method	Training	Generating $x_S^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
`Independence`	0.0	18.6	3:32.9	3:51.5	9.144
`Ctree`	18.1	18:26.6	29.4	19:14.1	1.424
`VAEAC-200`	4:13.4	15:47.1	5:16.6	25:17.1	1.467
`VAEAC-1000`	16:01.1	18:03.6	4:25.6	38:30.3	1.230
`VAEAC-10000`	2:41:54.8	15:58.2	5:31.9	3:03:24.9	1.194
`VAEAC-20000`	5:32:11.4	16:46.2	5:26.1	5:54:23.7	1.193
`VAEAC-40000`	11:19:30.9	17:24.9	11:26.3	11:48:22.1	**1.180**
`VAEAC-f-indir-200`	4:16.5	17:02.6	5:21.7	26:40.8	1.457
`VAEAC-f-indir-1000`	18:42.3	18:40.6	3:57.5	41:20.4	1.270
`VAEAC-f-indir-10000`	2:48:12.4	17:13.9	5:33.1	3:10:59.4	1.234
`VAEAC-f-indir-20000`	5:44:35.2	17:49.1	5:27.1	6:07:51.4	1.216
`VAEAC-f-indir-40000`	12:51:32.9	17:42.9	7:38.7	13:16:54.5	1.220
`VAEAC-f-dir-200`	4:16.5	17:02.6	0.0	21:19.1	1.761
`VAEAC-f-dir-1000`	18:42.3	18:40.6	0.0	37:22.9	1.322
`VAEAC-f-dir-10000`	2:48:12.4	17:13.9	0.0	3:05:26.3	1.268
`VAEAC-f-dir-20000`	5:44:35.2	17:49.1	0.0	6:02:24.3	1.239
`VAEAC-f-dir-40000`	12:51:32.9	17:42.9	0.3	13:09:16.1	1.264
`LM sep.`	0.6	—	0.3	0.9	1.581
`Poly-2 sep.`	2.7	—	0.4	3.1	1.338
`Poly-3 sep.`	3.2	—	0.5	3.7	1.314
`Poly-4 sep.`	3.7	—	0.8	4.5	1.306
`LM-inter-2 sep.`	0.9	—	0.3	1.2	1.381
`LM-inter-3 sep.`	1.4	—	0.3	1.7	1.357
`LM-inter-4 sep.`	2.1	—	0.4	2.5	1.360
`Poly-inter-2 sep.`	2.6	—	0.8	3.4	1.310
`Poly-inter-3 sep.`	4.5	—	0.1	4.6	1.512
`Poly-inter-4 sep.`	10.4	—	1.4	11.8	9.314
`GAM sep.`	56.3	—	13.4	1:09.7	1.299
`GAM-5 sep.`	22.7	—	2.1	24.8	1.302
`GAM-10 sep.`	36.2	—	2.2	38.4	1.297
`GAM-CV, sep.`	12:10.8	—	2.1	12:12.9	1.299
`PCR sep.`	11.6	—	0.4	12.0	1.788
`PLS sep.`	7.1	—	0.3	7.4	1.630
`PPR sep.`	3:33.5	—	1.1	3:34.6	1.185
`PPR-fixed sep.`	15.4	—	0.8	16.2	1.198
`KNN sep.`	1:02.8	—	8.7	1:11.5	1.366
`Tree sep.`	6.9	—	0.3	7.2	1.559
`RF sep.`	2:30:50.4	—	19.3	2:31:09.7	1.259
`RF-def sep.`	1:21.4	—	8.5	1:29.9	1.344
`CatBoost sep.`	18:24.7	—	0.3	18:25.0	1.213
`LM sur.`	5.9	—	1.3	7.2	2.770
`Poly-2 sur.`	7.8	—	1.5	9.3	2.625
`Poly-3 sur.`	7.7	—	1.5	9.2	2.577
`LM-inter-2 sur.`	27.4	—	1.9	29.3	1.705
`LM-inter-3 sur.`	9:11.8	—	5.2	9:17.0	1.443
`Poly-inter-2 sur.`	9.1	—	1.7	10.8	2.435
`Poly-inter-3 sur.`	29.5	—	3.3	32.8	1.929
`GAM sur.`	58.4	—	26.2	1:24.6	2.557
`GAM-5 sur.`	30.3	—	2.1	32.4	2.556
`GAM-10 sur.`	45.8	—	2.0	47.8	2.553
`PCR sur.`	46.2	—	0.6	46.8	2.818
`PLS sur.`	38.4	—	1.3	39.7	2.770
`PPR sur.`	55:28.4	—	3.2	55:31.6	1.538
`KNN sur.`	0.5	—	0.0	0.5	9.305
`Tree sur.`	25.7	—	0.6	26.3	2.914
`RF sur.`	3:45:58.8	—	35.3	3:46:34.1	1.311
`RF-def sur.`	4:25.1	—	12.0	4:37.1	1.473
`CatBoost sur.`	29:26.0	—	2.6	29:28.6	1.348
`NN-Frye-3000 sur.`	6:08:53.8	—	4.1	6:08:57.9	2.049
`NN-Frye-6000 sur.`	12:06:33.8	—	4.1	12:06:37.9	1.742
`NN-Frye-15000 sur.`	1:09:15:16.7	—	3.9	1:09:15:20.6	1.445
`NN-Frye-40000 sur.`	3:16:01:07.5	—	4.0	3:16:01:11.5	1.320
`NN-Frye-ES sur.`	4:53:28.7	—	6.4	4:53:35.1	1.973
`NN-Olsen-500 sur.`	2:21:43.4	—	3.2	2:21:46.6	1.282
`NN-Olsen-2500 sur.`	12:02:38.3	—	3.5	12:02:41.8	1.216
`NN-Olsen-10000 sur.`	2:01:07:27.4	—	2.9	2:01:07:30.3	1.192
`NN-Olsen-20000 sur.`	4:04:25:26.1	—	2.9	4:04:25:29.0	1.210
`NN-Olsen-ES sur.`	2:52:53.4	—	3.8	2:52:57.2	1.269

Table S3: $\text{Abalone}_{\text{all}}$ data set experiment: $M = 8$, $N_{\text{train}} = 3133$, and $N_{\text{test}} = 1044$.

Method	Training	Generating $\boldsymbol{x}_{\mathcal{S}}^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
`Independence`	0.0	7.8	30.6	38.4	0.196
`Empirical`	2.6	11.1	1.4	15.1	0.143
`Gaussian`	0.0	2:06.1	29.0	2:35.1	0.127
`Copula`	0.0	10:17.2	37.3	10:54.5	0.127
`GH`	4:34.8	2:29.6	26.9	7:31.3	0.133
`Ctree`	9.6	6:35.7	1.6	6:46.9	0.158
`VAEAC-200`	48.5	7:44.9	2:02.7	10:36.1	0.134
`VAEAC-1000`	2:35.1	7:04.5	1:41.1	11:20.7	0.131
`VAEAC-5000`	12:45.5	7:28.0	1:44.3	21:57.8	0.128
`VAEAC-10000`	21:37.3	8:38.8	1:23.8	31:39.9	0.128
`VAEAC-20000`	43:17.2	8:36.1	1:17.2	53:10.5	0.129
`VAEAC-f-indir-200`	41.0	7:55.9	1:41.8	10:18.7	0.137
`VAEAC-f-indir-1000`	2:41.6	7:33.1	1:44.7	11:59.4	0.137
`VAEAC-f-indir-5000`	13:33.1	7:53.8	1:44.1	23:11.0	0.133
`VAEAC-f-indir-10000`	22:53.1	9:17.1	1:29.4	33:39.6	0.134
`VAEAC-f-indir-20000`	44:32.1	8:58.6	1:09.5	54:40.2	0.130
`VAEAC-f-dir-200`	41.0	7:55.9	0.0	8:36.9	0.149
`VAEAC-f-dir-1000`	2:41.6	7:33.1	0.0	10:14.7	0.145
`VAEAC-f-dir-5000`	13:33.1	7:53.8	0.0	21:26.9	0.137
`VAEAC-f-dir-10000`	22:53.1	9:17.1	0.0	32:10.2	0.143
`VAEAC-f-dir-20000`	44:32.1	8:58.6	0.0	53:30.7	0.133
`LM sep.`	1.3	—	0.6	1.9	**0.126**
`LM-inter-2 sep.`	1.5	—	0.6	2.1	0.127
`LM-inter-3 sep.`	1.9	—	0.8	2.7	0.134
`LM-inter-4 sep.`	2.5	—	0.6	3.1	0.157
`Lasso sep.`	44.4	—	0.6	45.0	**0.126**
`Ridge sep.`	49.1	—	0.5	49.6	0.128
`Elastic sep.`	45.1	—	0.6	45.7	**0.126**
`GAM sep.`	59.2	—	4.4	1:03.6	**0.126**
`PCR sep.`	8.1	—	0.8	8.9	**0.126**
`PLS sep.`	7.0	—	0.7	7.7	**0.126**
`PPR sep.`	5:21.9	—	0.5	5:22.4	**0.126**
`PPR-fixed sep.`	8.2	—	0.1	8.3	0.145
`SVM sep.`	8.6	—	1.1	9.7	0.139
`KNN sep.`	17.3	—	3.0	20.3	0.150
`Tree sep.`	5.9	—	0.9	6.8	0.189
`RF sep.`	1:00:14.2	—	9.4	1:00:23.6	0.143
`RF-def sep.`	28.8	—	5.8	34.6	0.155
`XGBoost sep.`	1:10:10.8	—	0.9	1:10:11.7	0.137
`XGBoost-def sep.`	1:01.9	—	1.1	1:03.0	0.182
`CatBoost sep.`	18:40.3	—	0.3	18:40.6	0.135
`LM sur.`	3.6	—	0.8	4.4	0.165
`LM-inter-2 sur.`	19.3	—	1.1	20.4	0.134
`LM-inter-3 sur.`	11:51.0		2.7	11:53.7	0.128
`Lasso sur.`	6.9	—	0.2	7.1	0.165
`Ridge sur.`	7.9	—	0.2	8.1	0.165
`Elastic sur.`	5.6	—	0.4	6.0	0.165
`GAM sur.`	18.7	—	2.6	21.3	0.168
`PCR sur.`	22.7	—	0.7	23.4	0.165
`PLS sur.`	20.8	—	0.6	21.4	0.165
`PPR sur.`	4:11.9	—	1.1	4:13.0	0.136
`KNN sur.`	7.6	—	0.1	7.7	0.671
`Tree sur.`	15.3	—	0.8	16.1	0.254
`RF sur.`	1:33:37.6	—	13.7	1:33:51.3	0.143
`RF-def sur.`	2:28.8	—	7.6	2:36.4	0.159
`XGBoost sur.`	3:30:18.6	—	0.5	3:30:19.1	0.167
`XGBoost-def sur.`	1:11.1	—	0.6	1:11.7	0.202
`CatBoost sur.`	52.7	—	0.7	53.4	0.140
`NN-Frye-3000 sur.`	55:43.4	—	2.0	55:45.4	0.177
`NN-Frye-6000 sur.`	1:52:42.1	—	0.8	1:52:42.9	0.178
`NN-Frye-10000 sur.`	3:11:37.4	—	2.0	3:11:39.4	0.154
`NN-Frye-15000 sur.`	4:28:14.0	—	1.9	4:28:15.9	0.153
`NN-Frye-ES sur.`	35:12.8	—	1.6	35:14.4	0.191
`NN-Olsen-500 sur.`	17:39.8	—	1.2	17:41.0	0.136
`NN-Olsen-2500 sur.`	1:28:53.3	—	1.5	1:28:54.8	0.135
`NN-Olsen-7500 sur.`	4:27:26.2	—	1.5	4:27:27.7	0.139
`NN-Olsen-10000 sur.`	5:42:27.4	—	1.5	5:42:28.9	0.134
`NN-Olsen-ES sur.`	27:24.8	—	1.7	27:26.5	0.137

Table S4: Diabetes data set experiment: $M = 10$, $N_{\text{train}} = 332$, and $N_{\text{test}} = 110$.

Method	Training	Generating $\boldsymbol{x}_S^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
Independence	0.0	1:39.1	3:59:14.4	4:00:53.5	0.145
Empirical	1:54.3	4:22.9	2:27:01.3	2:33:18.5	0.088
Gaussian	0.0	11:06.4	3:57:22.2	4:08:28.6	0.118
Copula	0.0	54:01.1	3:59:35.5	4:53:36.6	0.107
GH	10:46.1	12:46.2	4:00:07.6	4:23:39.9	0.109
Burr	2:22.1	5:19.1	3:44:51.5	3:52:32.7	0.202
Ctree	1:06.3	33:35.5	27:59.7	1:02:41.5	0.102
VAEAC-200	1:51.6	36:22.8	3:59:54.5	4:38:08.9	0.103
VAEAC-1000	8:33.3	36:38.4	3:53:15.3	4:38:27.0	0.097
VAEAC-10000	1:34:59.6	40:42.5	3:53:55.8	6:09:37.9	0.093
VAEAC-20000	3:08:28.3	37:06.7	3:53:31.3	7:37:06.3	0.093
VAEAC-f-indir-200	2:10.4	40:23.6	3:59:21.4	4:41:55.4	0.105
VAEAC-f-indir-1000	9:55.4	40:43.6	3:49:50.5	4:40:29.5	0.096
VAEAC-f-indir-10000	1:33:53.4	43:21.2	4:00:27.9	6:17:42.5	0.098
VAEAC-f-indir-20000	3:17:31.3	43:09.3	4:00:22.9	8:01:03.5	0.097
VAEAC-f-dir-200	2:10.4	40:23.6	0.0	42:34.0	0.139
VAEAC-f-dir-1000	9:55.4	40:43.6	0.0	50:39.0	0.120
VAEAC-f-dir-10000	1:33:53.4	43:21.2	0.0	2:17:14.6	0.122
VAEAC-f-dir-20000	3:17:31.3	43:09.3	0.5	4:00:41.1	0.123
LM sep.	3.4	—	1.2	4.6	0.146
Poly-2 sep.	11.6	—	2.2	13.8	0.137
Poly-3 sep.	18.2	—	2.9	21.1	0.128
Poly-4 sep.	25.1	—	2.8	27.9	0.127
LM-inter-2 sep.	6.4	—	2.1	8.5	0.134
LM-inter-3 sep.	10.1	—	1.9	12.0	0.138
LM-inter-4 sep.	22.7	—	2.1	24.8	0.150
Poly-inter-2 sep.	17.4	—	6.1	23.5	0.128
Poly-inter-3 sep.	1:03.2	—	15.7	1:18.9	0.131
Poly-inter-4 sep.	4:43.3	—	1:21.1	6:04.4	1.142
Lasso sep.	2:00.1	—	1.3	2:01.4	0.146
Ridge sep.	2:44.1	—	1.5	2:45.6	0.147
Elastic sep.	2:33.0	—	1.7	2:34.7	0.146
GAM sep.	3:12.0	—	12.6	3:24.6	0.124
GAM-5 sep.	40.8	—	19.6	1:00.4	0.124
GAM-10 sep.	41.2	—	17.9	59.1	0.122
GAM-CV, sep.	31:29.5	—	16.9	31:46.4	0.124
PCR sep.	1:04.2	—	3.2	1:07.4	0.146
PLS sep.	53.3	—	2.5	55.8	0.146
PPR sep.	25:17.3	—	2.1	25:19.4	0.129
PPR sep.	25:17.3	—	2.1	25:19.4	0.129
PPR-fixed sep.	54.3	—	3.2	57.5	0.130
SVM sep.	3:42.3	—	12.2	3:54.5	0.109
KNN sep.	3:37.1	—	21.3	3:58.4	0.121
Tree sep.	48.7	—	2.7	51.4	0.132
RF sep.	9:20:25.2	—	47.7	9:21:12.9	**0.071**
RF-def sep.	4:00.4	—	23.3	4:23.7	0.072
XGBoost-def sep.	4:32.5	—	3.2	4:35.7	0.103
CatBoost sep.	1:41:31.6	—	1.1	1:41:32.7	0.082
LM sur.	22.5	—	3.4	25.9	0.162
Poly-2 sur.	39.0	—	4.7	43.7	0.155
Poly-3 sur.	43.7	—	6.2	49.9	0.147
Lasso sur.	1:11.2	—	0.4	1:11.6	0.167
Ridge sur.	1:30.2	—	1.0	1:31.2	0.182
Elastic sur.	1:23.7	—	1.3	1:25.0	0.166
GAM sur.	3:38.5	—	22.9	4:01.4	0.145
GAM-5 sur.	11:29.9	—	10.3	11:40.2	0.144
GAM-10 sur.	11:15.0	—	13.8	11:28.8	0.142
PCR sur.	4:37.8	—	4.3	4:42.1	0.162
PLS sur.	4:34.8	—	5.7	4:40.5	0.162
PPR sur.	1:20:09.6	—	5.9	1:20:15.5	0.149
KNN sur.	31.0	—	0.2	31.2	0.369
Tree sur.	3:40.8	—	3.7	3:44.5	0.214
RF sur.	1:15:41:45.0	—	1:01.3	1:15:42:46.3	0.085
RF-def sur.	27:59.0	—	42.0	28:41.0	0.075
XGBoost-def sur.	15:53.3	—	3.6	15:56.9	0.113
CatBoost sur.	29:01.9	—	4.5	29:06.4	0.108
NN-Frye-3000 sur.	1:43:10.9	—	6.3	1:43:17.2	0.227
NN-Frye-6000 sur.	3:22:15.2	—	6.2	3:22:21.4	0.210
NN-Frye-15000 sur.	17:35:44.9	—	8.2	17:35:53.1	0.190
NN-Frye-40000 sur.	1:10:53:20.6	—	6.8	1:10:53:27.4	0.170
NN-Frye-ES sur.	1:39:33.7	—	8.6	1:39:42.3	0.272
NN-Olsen-500 sur.	35:04.3	—	5.1	35:09.4	0.145
NN-Olsen-2500 sur.	2:55:22.9	—	5.4	2:55:28.3	0.137
NN-Olsen-10000 sur.	23:56:39.8	—	7.2	23:56:47.0	0.130
NN-Olsen-20000 sur.	1:23:54:17.3	—	6.2	1:23:54:23.5	0.132
NN-Olsen-ES sur.	2:42:45.9	—	9.0	2:42:54.9	0.116

Table S5: (Red) Wine data set experiment: $M = 11$, $N_{\text{train}} = 1349$, and $N_{\text{test}} = 250$.

Method	Training	Generating $\boldsymbol{x}_S^{(k)}$	Predicting $v(\mathcal{S})$	Total CPU Time	MSE_v
`Independence`	0.0	34:31.9	42:14.4	1:16:46.3	0.041
`VAEAC-200`	1:05:55:35.9	4:05:53:10.4	42:17.4	5:12:31:03.7	0.027
`VAEAC-1000`	6:01:10:13.0	3:23:34:03.1	51:35.8	10:01:35:51.9	0.027
`VAEAC-f-indir-200`	1:10:59:08.6	5:19:33:00.1	1:01:06.4	7:07:33:15.1	0.027
`VAEAC-f-indir-1000`	6:17:32:29.9	4:22:48:45.9	53:42.3	11:17:14:58.1	0.027
`VAEAC-f-dir-200`	1:10:59:08.6	5:19:33:00.1	0.2	7:06:32:08.9	0.033
`VAEAC-f-dir-1000`	6:17:32:29.9	4:22:48:45.9	0.3	11:16:21:16.1	0.032
`LM sep.`	8:38:57.2	—	21:16.3	9:00:13.5	0.043
`Poly-2 sep.`	13:44:03.3	—	27:29.3	14:11:32.6	0.037
`Poly-3 sep.`	17:00:32.0	—	29:08.1	17:29:40.1	0.037
`Poly-4 sep.`	19:25:17.4	—	28:26.1	19:53:43.5	0.036
`Poly-inter-2 sep.`	11:53:01.3	—	29:00.7	12:22:02.0	0.036
`GAM sep.`	2:37:14.8	—	1:37.8	2:38:52.6	0.033
`GAM-5 sep.`	7:19:07:43.1	—	2:12:59.1	7:21:20:42.2	0.034
`GAM-10 sep.`	75:22:18:27.1	—	2:05:52.8	76:00:24:19.9	0.033
`PLS sep.`	25:04:43:30.2	—	17:21.1	25:05:00:51.3	0.046
`PPR sep.`	14:12:16:08.6	—	3:39.0	14:12:19:47.6	0.032
`PPR-fixed sep.`	1:15:44:05.5	—	3:41.7	1:15:47:47.2	0.032
`Tree sep.`	5:44:54.7	—	46.0	5:45:40.7	0.031
`RF sep.`	98:13:17:15.8	—	16:11.6	98:13:33:27.4	0.027
`RF-def sep.`	12:49:17.0	—	8:39.0	12:57:56.0	0.028
`CatBoost sep.`	35:09:59:20.8	—	38.3	35:09:59:59.1	**0.026**
`NN-Frye-3000 sur.`	3:16:10:45.4	—	3:04.9	3:16:13:50.3	0.085
`NN-Frye-15000 sur.`	81:00:01:50.5	—	3:02.3	81:00:04:52.8	0.098
`NN-Frye-ES sur.`	9:42:32.0	—	2:24.8	9:44:56.8	0.101
`NN-Olsen-500 sur.`	3:11:31:47.6	—	2:10.3	3:11:33:57.9	0.045
`NN-Olsen-2500 sur.`	14:17:59:53.4	—	2:19.0	14:18:02:12.4	0.065
`NN-Olsen-10000 sur.`	82:11:56:11.3	—	2:36.9	82:11:58:48.2	0.037
`NN-Olsen-ES sur.`	1:13:15:31.9	—	1:58.4	1:13:17:30.3	0.030

Table S6: The Adult data set experiment: $M = 14$, $N_{\text{train}} = 30\,000$, and $N_{\text{test}} = 162$. Creating the augmented test data on the form in (5) takes approximately two minutes and is part of the predicting time for the `NN surrogate` approaches.

S5 Schematic Overview of Conditional Shapley Values in XAI

Figure S11 provides a schematic overview of this article's method classes and methods for computing conditional Shapley value explanations. Furthermore, the figure also shows conditional Shapley values' place within the explainable artificial intelligence field as a model-agnostic explanation framework with local explanations. Note that the ellipses represent the additional methods introduced in Section S1. Furthermore, LIME is an explanation framework developed by Ribeiro et al. (2016), and see, e.g., Molnar (2022)[Section 9.3] and Redelmeier et al. (2021) for more on counterfactual explanations.

In this article, we use Shapley values to provide local explanations for models fitted to tabular data, but different Shapley value-based frameworks are developed for other settings. For example, Lundberg et al. (2020) develop model-specific local and global Shapley value explanations for tree ensemble models. Covert et al. (2020) introduce a model-agnostic global explanation framework based on Shapley values. Krzyziński et al. (2023) propose time-dependent Shapley value-based explanations for machine learning survival models. Bento et al. (2021) extend Shapley values to the sequential domain and introduce a model-agnostic Shapley value explanation framework for sequential decision-making models, such as recurrent neural networks. Chen et al. (2022b) explain a series of models by propagating Shapley values. Jethani et al. (2021) use Shapley values to explain classifications made by image classifiers. Duval and Malliaros (2021); Mastropietro et al. (2022) introduce new methodologies based on Shapley values to explain graph neural network models. Wang et al. (2021) include knowledge about a causal graph between the features when creating Shapley value-based explanations. Heskes et al. (2020) also introduces a causal Shapley value methodology that exploits causal knowledge.

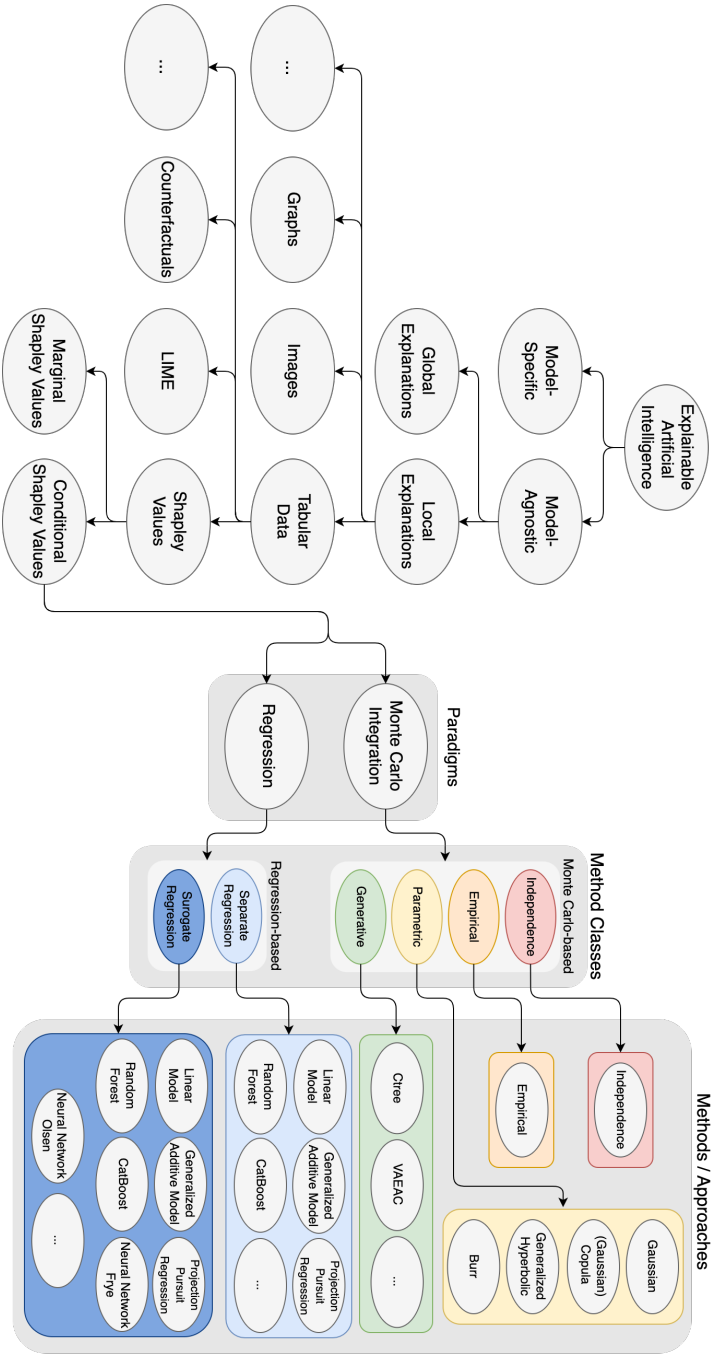

Fig. S11: Schematic overview of conditional Shapley values within XAI.

References

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

Belghazi M, Oquab M, Lopez-Paz D (2019) Learning about an exponential amount of conditional distributions. In: Wallach H, Larochelle H, Beygelzimer A, et al. (eds) Advances in Neural Information Processing Systems, vol 32. Curran Associates, Inc.

Bénard C, Biau G, Da Veiga S, et al. (2022) Shaff: Fast and consistent shapley effect estimates via random forests. In: Camps-Valls G, Ruiz FJR, Valera I (eds) Proceedings of The 25th International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, vol 151. PMLR, pp 5563–5582, URL https://proceedings.mlr.press/v151/benard22a.html

Bento J, Saleiro P, Cruz AF, et al. (2021) Timeshap: Explaining recurrent models through sequence perturbations. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp 2565–2573

Bergmeir C, Benítez JM (2012) Neural networks in R using the stuttgart neural network simulator: RSNNS. Journal of Statistical Software 46(7):1–26. URL https://www.jstatsoft.org/v46/i07/

Chen H, Covert IC, Lundberg SM, et al. (2022a) Algorithms to estimate shapley value feature attributions. arXiv preprint arXiv:220707605

Chen H, Lundberg SM, Lee SI (2022b) Explaining a series of models by propagating shapley values. Nature communications 13(1):4512

Chen T, He T, Benesty M, et al. (2015) Xgboost: extreme gradient boosting. R package version 04-2 1(4):1–4

Covert I, Lundberg SM, Lee SI (2020) Understanding global feature contributions with additive importance measures. Advances in Neural Information Processing Systems 33

Covert I, Lundberg S, Lee SI (2021) Explaining by removing: A unified framework for model explanation. Journal of Machine Learning Research 22(209):1–90

Douglas L, Zarov I, Gourgoulias K, et al. (2017) A universal marginalizer for amortized inference in generative models. In: Proceedings of 31st Conference on Neural Information Processing Systems (NIPS 2017)

Duval A, Malliaros FD (2021) Graphsvx: Shapley value explanations for graph neural networks. In: Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21, Springer, pp 302–318

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

Friedman JH, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software 33(1):1–22. https://doi.org/10.18637/jss.v033.i01, URL https://www.jstatsoft.org/index.php/jss/article/view/v033i01

Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, pp 249–256

Gondara L, Wang K (2018) Mida: Multiple imputation using denoising autoencoders. In: Phung D, Tseng VS, Webb GI, et al. (eds) Advances in Knowledge Discovery and Data Mining. Springer International Publishing, Cham, pp 260–272

Hastie T (2022) gam: Generalized Additive Models. URL https://CRAN.R-project.org/package=gam, r package version 1.20.1

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

He K, Zhang X, Ren S, et al. (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp 1026–1034

Heskes T, Sijben E, Bucur IG, et al. (2020) Causal shapley values: Exploiting causal knowledge to explain individual predictions of complex models. Advances in Neural Information Processing Systems 33

Ivanov O, Figurnov M, Vetrov D (2019) Variational autoencoder with arbitrary conditioning. In: International Conference on Learning Representations

Jethani N, Sudarshan M, Covert IC, et al. (2021) Fastshap: Real-time shapley value estimation. In: International Conference on Learning Representations

Krzyziński M, Spytek M, Baniecki H, et al. (2023) Survshap(t): Time-dependent explanations of machine learning survival models. Knowledge-Based Systems 262:110,234. https://doi.org/https://doi.org/10.1016/j.knosys.2022.110234, URL https://www.sciencedirect.com/science/article/pii/S0950705122013302

Li Y, Akbar S, Oliva J (2020) Acflow: Flow models for arbitrary conditional likelihoods. In: International Conference on Machine Learning, PMLR, pp 5831–5841

Liland KH, Mevik BH, Wehrens R (2021) pls: Partial Least Squares and Principal Component Regression. URL https://CRAN.R-project.org/package=pls, r package version 2.8-0

Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. In: Advances in neural information processing systems, pp 4765–4774

Lundberg SM, Erion G, Chen H, et al. (2020) From local explanations to global understanding with explainable ai for trees. Nature machine intelligence 2(1):56–67

Mastropietro A, Pasculli G, Feldmann C, et al. (2022) Edgeshaper: Bond-centric shapley value-based explanation method for graph neural networks. Iscience 25(10):105,043

Mattei PA, Frellsen J (2019) Miwae: Deep generative modelling and imputation of incomplete data sets. In: International conference on machine learning, PMLR, pp 4413–4423

Meyer D, Dimitriadou E, Hornik K, et al. (2022) e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. URL https://CRAN.R-project.org/package=e1071, r package version 1.7-12

Molnar C (2022) Interpretable Machine Learning, 2nd edn. URL https://christophm.github.io/interpretable-ml-book

Olsen LHB, Glad IK, Jullum M, et al. (2022) Using shapley values and variational autoencoders to explain predictive models with dependent mixed features. Journal of Machine Learning Research 23(213):1–51

Prokhorenkova L, Gusev G, Vorobev A, et al. (2018) Catboost: unbiased boosting with categorical features. In: Bengio S, Wallach H, Larochelle H, et al. (eds) Advances in Neural Information Processing Systems, vol 31. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf

Redelmeier A, Jullum M, Aas K, et al. (2021) Mcce: Monte carlo sampling of realistic counterfactual explanations. arXiv preprint arXiv:211109790

Ribeiro MT, Singh S, Guestrin C (2016) " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1135–1144

Schliep K, Hechenbichler K (2016) kknn: Weighted k-Nearest Neighbors. URL https://CRAN.R-project.org/package=kknn, r package version 1.3.1

Stekhoven DJ, Bühlmann P (2011) MissForest—non-parametric missing value imputation for mixed-type data. Bioinformatics 28(1):112–118. https://doi.org/10.1093/bioinformatics/btr597

Therneau T, Atkinson B (2022) rpart: Recursive Partitioning and Regression Trees. URL https://CRAN.R-project.org/package=rpart, r package version 4.1.16

Uria B, Côté MA, Gregor K, et al. (2016) Neural autoregressive distribution estimation. The Journal of Machine Learning Research 17(1):7184–7220

Van Buuren S, Groothuis-Oudshoorn K (2011) mice: Multivariate imputation by chained equations in r. Journal of statistical software 45:1–67

Wang J, Wiens J, Lundberg S (2021) Shapley flow: A graph-based approach to interpreting model predictions. In: International Conference on Artificial Intelligence and Statistics, PMLR, pp 721–729

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

Xu T, Chang CC, Lin CC, et al. (2021) WeightSVM: Subject Weighted Support Vector Machines. URL https://CRAN.R-project.org/package=WeightSVM, r package version 1.7-9

Yoon J, Jordon J, Schaar M (2018) Gain: Missing data imputation using generative adversarial nets. In: International Conference on Machine Learning, PMLR, pp 5689–5698

Zheng S, Charoenphakdee N (2022) Diffusion models for missing value imputation in tabular data. arXiv preprint arXiv:221017128