Université de Neuchâtel
Faculté des sciences

Explainable Machine Learning: Approximating Shapley Values for Dependent Predictors

Thesis submitted in partial fulfillment of the requirements for the degree

Master of Science in Statistics

by

Jan Kasperek

January 2024

Supervisor:
Prof. tit. Dr. Alina Matei
Université de Neuchâtel
Faculty of Science
Institute of Statistics

Abstract

Modern Machine Learning algorithms often outperform classical statistical methods in predictive accuracy. This comes at the expense of model interpretability. As businesses and institutions increasingly rely on Machine Learning to support and automate decision making processes to reap the benefits of more accurate predictions, explaining these model outputs becomes more important. A universally applicable approach to explaining such complex models is based on the Shapley value, a concept originating from game theory. However, its calculation is very computer-intensive, so approximations have to be used. The state-of-the-art approach, *Kernel SHAP*, assumes independence of the predictors, which is unrealistic in practice. Recent research has developed improvements to incorporate dependencies between predictors. After a review of the theoretical underpinnings, the original KernelSHAP method is compared with improved versions in realistic settings, using three real-world datasets. While the improved versions are found to have smaller approximation error to exact Shapley values, they are also more computationally demanding. Further improvements are discussed and possible research directions are suggested. The thesis is structured as follows: After introducing explainable machine learning in chapter 1, the Shapley value and its applications to model explainability are explored in chapter 2. Chapter 3 presents methods to approximate Shapley values as well as recent improvements to these methods, which are tested on real datasets in chapter 4. Some possible directions for future research are pointed out in chapter 5, before giving a final conclusion in chapter 6. Code for the experiments of chapter 4 is found in the appendix.

Contents

1. Introduction: Explainable Machine Learning

With ever-increasing computational resources at decreasing costs, the use of complex predictive models becomes more widespread. Such machine learning models often offer an advantage when it comes to predictive accuracy, making it an attractive choice for prediction problems in enterprises and other institutions over classical statistical models. However, higher model complexity comes at the cost of lower interpretability: It becomes less clear how the model inputs (predictors) actually relate to the output (i.e., the predictions). For instance, in a classical linear regression model, it is immediately visible how any given predictor affects the predictions: Both are linked by a simple linear function. Such a simple link does not exist for many other commonly used predictive models such as neural networks or tree ensembles such as random forests.

Since such "black-box" machine learning models are increasingly being used to aid critical decision-making due to their predictive qualities, there is a demand to make these models explainable. That is, to provide an explanation what lead the model to its predictions, explicitly linking predictors (independent variables) to predictions and thus making these models more transparent. For instance, banks automate their decisions on loan applications by using machine learning models, saving costs by replacing credit analysts. In the medical field, clinical decision support software is deployed to aid medical experts in giving the correct diagnosis. Large enterprises may partly automate their hiring process by automatically pre-selecting candidates using artificial intelligence (AI). Such software may give recommendations based on complex predictive models. In all examples, the fully or partially automated decision is based on personal data of the client and can have important implications for this person. Thus, ethical and legal questions arise: Does the decision process (and thereby the model) fulfill certain notions of fairness? Is it compliant with current laws and regulations? With more widespread usage of machine learning in various sectors, regulators seem increasingly committed in ensuring that decisions based on such models are explainable. This is not only due to the direct impact on individuals; concern is also raised at the macro level, as is the case for the financial system. For instance: In its 2023 risk monitor report [1], the Swiss Financial Market Supervisory Authority (FINMA) dedicated a seperate section on the use of artificial intelligence in the swiss financial sector, stressing the importance of explainability.

Explainable machine learning can also be of use to modelers for diagnostic purposes, detecting weaknesses and assessing the plausibility of estimated associations between predictors and the target variable. Moreover, models are being developed with help of domain experts (e.g. when it comes to variable selection) who may not be well-versed in statistics and machine learning. Simple model explanations may thus aid the process of building models which are more useful for the problem at hand. Simple explanations can also increase trust in a complex model, which is crucial when stakeholders decide on its deployment. If decision makers, clients and other agents possibly affected by a model's future predictions are not convinced by its usefulness, the model may not be deployed at all, making all efforts put into its development futile.

[1] https://www.finma.ch/en/~/media/finma/dokumente/dokumentencenter/myfinma/finma-publikationen/risikomonitor/20231109-finma-risikomonitor-2023.pdf?sc_lang=en&hash=27012208F65AFDAC00DBE199ADF8DB40 (retrieved: Nov 18, 2023)

Generally, we will deal with supervised models, i.e. problems where there exists a target variable. While unsupervised models (e.g. clustering) also have practical use cases such as customer segmentation or fraud detection, such models are often not directly used to make high-stakes decisions. However, there are approaches being developed for explainable unsupervised machine learning as well (see for instance Montavon et al. (2022)). Moreover, in some cases one could convert an unsupervised problem into a supervised one (for example by assigning class labels to clusters found by an unsupervised algorithm). To limit ourselves to the most common use cases, we will also only focus on explainability of models for cross-sectional data where the focus is on prediction and not inference. This excludes model classes such as mixed models or time series analysis from what follows. An important distinction can also be made between *local* and *global* explanations. While the latter makes statements about the average or typical behavior of a model, local methods aim to explain predictions for single observations or groups of similar observations. Since complex models such as tree-based ensembles may automatically capture interactions, non-linear or even non-monotonic relationships between predictors, local explanations cannot be deduced from global ones in general, but the converse is possible: One can aggregate local explanations to global ones, for example by averaging over observations. One such example will be given in Section 1.1.1. Therefore, local explanation methods generally offer more possibilities and a more granular look into a model's behavior; the focus of this work is thus on these approaches. Another distinction between explainability methods, *model-specific* vs *model-agnostic*, is the topic of the following sections, where some example methods will be given. For an extensive survey of explainable supervised machine learning refer to Burkart & Huber (2021).

1.1 Model-Specific Approaches

An important distinction of explainability methods is made between *model-specific* and *model-agnostic* approaches. While this thesis deals with the latter, it is still instructive to discuss model-specific approaches since contrasting these methods with model-agnostic approaches helps making clear the benefits of the latter. Model-specific explainability methods only work for a certain model type, i.e., there exist approaches specifically for regression, tree-based models, neural networks, and many other. These approaches exploit properties of the model they were designed for in order to generate local or global explanations.

Examples include Breiman (2001), who in his seminal paper, alongside his invention of random forests, proposed a measure of variable importance for this model. It is based on changing observed values of the predictor in question and then recording the increase in prediction error; predictors with a larger increase are deemed more influential on the model predictions. Another possibility to measure global predictor importance in tree-based models consists of counting the number of tree splits in which a given predictor was used. For linear regression models, Grömping (2015) offers an extensive discussion of global explanations (*variable importance*). To give a practical example of a *local* model-specific explanation method, one method is outlined for a class of regression models in the following subsection. This will also introduce a model which will be applied later on in this thesis.

1.1.1 Example: Generalized Linear Models

Introduced by Nelder & McCullagh (1983), Generalized Linear Models (GLM's) can be seen as a generalization of the linear regression model, offering a unifying definition and estimation approach to several different parametric regression models. We will consider the predictors x_j, $j \in \{1, ...k\}$ as fixed. Y represents the target variable and $\beta_0, ..., \beta_k$ are the parameters of interest to be estimated. The linear regression model can then be written as a linear combination

$$\mathbb{E}(Y_i \mid x_{i1}, ..., x_{ik}) = \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij}$$

where $i = \{1, ..., n\}$ represents the independent observations and $j = \{1, ..., k\}$ indexes the predictors with the associated parameters β_j.

The model extension is the following: Let Y be distributed according to an exponential family. On the left-hand side of the model equation, we now consider a certain transformation of the target variable Y through a monotonic *link function* $g()$ (with a differentiable inverse), so the model becomes

$$g(\mathbb{E}(Y_i \mid x_{i1}, ..., x_{ik})) = \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij} \tag{1}$$

On the scale of the target variable, a GLM is thus written

$$\mathbb{E}(Y_i \mid x_{i1}, ..., x_{ik}) = g^{-1}(\beta_0 + \sum_{j=1}^{k} \beta_j x_{ij})$$

The use of $g()$ insures that all predicted values stay in an admissible range. For instance, a categorical target variable should naturally be predicted by values between 0 and 1 so the predicted values can be interpreted as class probabilities. Similarly, counts or waiting times cannot take negative values. A linear regression model might deliver predictions outside the admissible ranges for all of these cases. Moreover, this model assumes the absence of any non-linear and multiplicative effects, features that can be incorporated into a GLM by an appropriate choice of $g()$.

However, a linear regression model is much more intuitive to interpret. Ceteris paribus, a unit increase in a predictor x_j induces a change of β_j in the predicted value of Y, no matter the value of x_j and on the scale of the target variable itself. This is not the case in GLM's if $g()$ is not the identity function. Thus, different observations will have varying partial effects. Or, in other words: The partial derivatives with respect to a parameter of interest β_j of the regression function is constant in the linear regression case, but varies otherwise. While the parameters may still have a clearly defined interpretation, it is not as intuitive to comprehend. This motivates the following generalization for the estimation of partial effects in GLM's, similar formulations of which are frequently found in the econometrics literature (see for instance Wooldridge (2010)). However, similar explanation methods based on gradients have also appeared in the machine learning and artificial intelligence literature, see Baehrens et al. (2010) and Sundararajan et al. (2017).

Restricting our attention to continuous predictors, we define the partial effect (PE) of predictor x_j at observation i as

$$PE_i(x_j) = \frac{\partial g^{-1}(\beta_0 + \sum_{j=1}^{k} \beta_j x_{ij})}{\partial x_j}$$

That is, increasing a predictor x_j by one (small) unit is estimated to induce a change of $PE_i(x_j)$ in the target variable Y_i for the observational unit i. To obtain a similar interpretation as in linear regression while retaining the higher flexibility of a GLM, we may average the observation-specific partial effects, yielding so-called Average Partial Effects (APE) for each predictor. For categorical predictors, one can consider the differences between two levels of interest instead of partial derivatives. Partial effects for interaction terms can be computed by numerical methods. Additionally, a closed-form expression for standard errors of APE's is readily available via the Delta-method. For a more detailed description and software implementation refer to Leeper (2017).

While this method based on partial derivatives may work well for GLM's, some models make predictions by estimating non-differentiable functions (e.g. tree-based algorithms). Others have a more complex structure in general (for instance neural networks). Since it is common to utilize various models, a general, *model-agnostic* approach to explainability seems desirable.

1.2 Model-Agnostic Approaches

Model-agnostic approaches to interpretable machine learning seek to provide explanations for the whole set of predictive supervised models. The machine learning and artificial intelligence fields are developing rapidly and many different models have found widespread use. Hence, it is not uncommon for companies and other organizations to employ multiple different models and possibly utilize newly developed ones within a short period of time. It is also common practice to try out different models when developing systems based on machine learning algorithms. Thus, model-agnostic approaches are advantageous for many use cases: Comparing the behavior of different models in the development phase, increasing trust of decision makers (who are not experts in the area of machine learning) in the model in order to get it deployed, as well as explaining decisions made with the aid of (possibly) multiple different models which may change over time.

A conceptually simple way of making complex models interpretable is to fit a *surrogate model* after the machine learning algorithm has been trained. A surrogate is a model that is easily interpretable by itself, such as linear regression or a single decision tree (for example, Breiman et al. (1984)). This surrogate is fitted on the predictions of a machine learning algorithm as the target variable, while the predictors remain the same. Let $\hat{y_i}^{ML}$ be the prediction of a complex machine learning model. With a simple function $h()$, and predictors $x_j, j \in \{1, ..., k\}$ a surrogate model can then be written

$$\hat{y_i}^{ML} = h(x_{i1}, ..., x_{ik}), \forall i \in \{1, ..., n\}$$

Critique can be directed at the fact that surrogate models are not fully model-agnostic, since an appropriate form for h has to be chosen, which can differ across machine learning models (for example, along the lines of classification versus regression). A prominent example of

a surrogate model framework is LIME (Ribeiro et al., 2016), which fits simple models (e.g., linear) locally around $\hat{y_i}^{ML}$, the prediction of interest. To control surrogate model complexity and avoid its overfitting, a penalty term is introduced in the objective function of the surrogate and an appropriate bandwidth to define the neighbourhood of any $\hat{y_i}^{ML}$ has to be chosen.

A graphical way to detect the influence a given predictor has on a model prediction for some unit $i \in \{1, ...n\}$ are individual conditional expectation (ICE) plots (Goldstein et al., 2015). ICE plots are constructed by expanding an equivalent method, *partial dependence plots*, for detecting global dependencies popularized by Friedman (2001) to obtaining local explanations: For each observational unit i, the values of a given predictor x_j are changed across its empirical range. The ICE plot then displays the relationship between different hypothetical values of x_j and the predictions that result from plugging in those values while holding all other predictors fixed. Partial dependence plots can then be obtained by simply averaging over all individual curves. While easy to interpret, this method ignores any dependencies between the predictors by only considering the marginal predictor distributions. Thus, certain ranges shown in an ICE plot may be unrealistic with respect to the joint distribution of predictors; holding all other predictors fixed while generating realistic datapoints may only work in a small neighbourhood of the observed predictor values.

This is an issue with most *permutation-based* approaches: Changing observed predictor values only according to their marginal distribution is tempting due to the simplicity of this approach; simulating data from an estimated joint distribution of predictors is challenging, especially in high dimensions. But as soon as there are considerable dependencies between predictors, misleading explanations may be generated.

Another approach is to remove a predictor of interest from the model, re-compute predictions without this variable in the model and observe the difference between predictions. Like permutation-based methods, this does not take into account interactions between predictors that can be detected by models like tree-based ensembles. If the number of predictors is sufficiently small, it might be feasible to fit models for all possible predictor combinations in order to average over these combinations. In fact, the concept to which this thesis is dedicated follows a similar approach. However, approximations are necessary when the number of predictors grows. One way to deal with this problem is to simulate absence of predictors by setting them to "non-informative" values (see for instance Robnik-Sikonja & Kononenko (2008) for such techniques), which brings back permutation-based methods and their drawbacks.

Like local versus global methods, model-agnostic approaches offer more flexibility and generality compared to their model-dependent alternatives. Finding methods capable of providing both local and model-agnostic explanations thus seems worthwhile. Another reasonable criterion for a good explanation method is that it gives attributions to predictors on the scale of the target variable and in an additive fashion, so its outputs are easily comprehensible to a wide audience. The most widespread methods satisfying all these criteria turn out to be based on the *Shapley value* (to which the next chapter is dedicated), which offers a sound theoretical basis that other approaches often lack.

2. The Shapley Value

The Shapley value is a concept from cooperative game theory. Originally introduced by Shapley (1953), it offers a unique solution to cooperative games with attractive properties for applications in explainability. Most importantly, it decomposes model predictions in an additive fashion while taking account of all possible predictor interactions. The use of the Shapley value to explain predictions of complex, "black-box" machine learning algorithms was introduced by Štrumbelj & Kononenko (2010). Lundberg & Lee (2017) proposed a unifying framework to connect the Shapley value with other existing explainability methods, while also introducing *KernelSHAP*, a computationally efficient method for its approximation. Since this method relies on assumptions regarding the independence between predictors, the approximations of KernelSHAP can be inaccurate in commonly encountered scenarios. To tackle this issue, Aas, Jullum, et al. (2021) developed a method which incorporates dependencies between predictors. This chapter lays the theoretical foundation for the application of Shapley value based techniques in explainable machine learning, starting with the original game-theoretic concept.

2.1 The Shapley Value in Game Theory

Game theory is the mathematical study of strategic decision-making when multiple agents with possibly conflicting incentives are involved. Under certain assumptions of individual rationality and utility maximization, interactions of different agents can be mathematically modeled as a *game* G with a set of actions each player $j \in \mathcal{A} = \{1, ..., k\}$ can take at a given stage. Moreover, a game is characterized by a payoff function, which assigns payoffs (or *gains*) to the players for each possible combination of their actions. Given this setting, one can then find optimal strategies for players and describe equilibria, i.e. assign strategies to each player such that no agent has an incentive to deviate from his course of action. One can differentiate between *cooperative* and *non-cooperative* games; the latter is the more common setting in game theory. In non-cooperative games, each agent seeks to maximize his own utility and acts as an independent unit working for his own payoff; cooperation between players is not externally enforced. Cooperative game theory describes settings where multiple agents act together to achieve a collective payoff; cooperation can be insured by binding rules and external enforcement. Here, game theory may predict collective action or find equilibria in the sense of "fair" payoff distributions among the group members. This is a task tackled by the Shapley value (Shapley, 1953), which is later adopted for explaining predictions of complex, "black-box" machine learning models. In general, game theory has applications mainly in the social sciences (particularly economics and political science), but also in computer science (Nisan & Ronen, 2001) and biology (Smith & Price, 1973).

In the setting of cooperative games, the Shapley value (Shapley, 1953) is a solution to the problem of how to divide a collective payoff among players. This solution is attractive because it is unique in fulfilling certain axioms characterizing a "fair" way of distributing the group gains. In what follows, the key concepts and axioms leading to the Shapley value are introduced.

A cooperative game G can be written $G(\mathcal{A}, v)$ where $\mathcal{A} = \{1, 2, ..., k\}$, $k \geqslant 2$ denotes the set of all players and $v : \mathcal{P}(\mathcal{A}) \mapsto \mathbb{R}$ with $v(\{\emptyset\}) = 0$ is the so-called *characteristic function* or *coalition*

valuation of the game. ν assigns a joint payoff to each possible coalition of players ($\mathcal{P}(\mathcal{A})$ denotes the power set of $\mathcal{A}$), with the empty coalition receiving zero payoff. It is assumed that gains made by any coalition can be split freely among its members. For the following we presuppose that all payoffs received by single players depend only on ν. We now search for a payoff vector $\boldsymbol{\phi}(G)$ with entries $\phi_j, j \in \mathcal{A}$, assigning to each player a "fair" payoff for his contribution to the joint gains. A "fair" distribution can be characterized by the following axioms:

(A1) Efficiency: The sum of all player's payoffs must be equal to the joint payoff of the coalition including all players. The whole coalition valuation is spread among its members.

$$\sum_{j=1}^{k} \phi_j = \nu(\mathcal{A})$$

(A2) Dummy player: A player who does not contribute to the payoff of any possible coalition C, should not get a payoff.

$$\nu(C \setminus \{j\}) = \nu(C), \forall C \subseteq \mathcal{A} \Rightarrow \phi_j = 0$$

(A3) Symmetry: If two players $\{j, l\}$ contribute the same to all possible coalitions, they should receive the same payoff.

$$\nu(D \cup \{j\}) = \nu(D \cup \{l\}), \forall D \subseteq (\mathcal{A} \setminus \{j, l\}) \Rightarrow \phi_j = \phi_l$$

(A4) Additivity & Linearity: For a set of cooperative games $\{G_1, G_2, G_3\}$ with the same set of players $\mathcal{A}$ $G_1(\mathcal{A}, \nu_1)$, $G_2(\mathcal{A}, \nu_2)$, and $G_3(\mathcal{A}, \nu_3 = \nu_1 + \nu_2)$,

$$\boldsymbol{\phi}(G_1) + \boldsymbol{\phi}(G_2) = \boldsymbol{\phi}(G_3)$$

and, for any ν:

$$\boldsymbol{\phi}(\lambda \cdot \nu) = \lambda \cdot \boldsymbol{\phi}(\nu), \forall \lambda \in \mathbb{R}$$

If the coalition valuation of one game is the sum of coalition valuations from other games, the payoffs should be the sums of gains from these other games. Moreover, multiplying the joint payoff ν by a scalar yields individual payoffs equal to the payoffs given by ν times that scalar.

Shapley (1953) showed that for any cooperative game G there exists a unique solution for $\boldsymbol{\phi}(G)$ fulfilling (A1) to (A4) under the assumptions stated before. This solution, referred to as the *Shapley Value*, is given for any player $j \in \mathcal{A}$ by

$$\phi_j(\nu) = \frac{1}{k!} \sum_{B \subseteq \mathcal{A} \setminus \{j\}} |B|!(k - |B| - 1)![\nu(B \cup \{j\}) - \nu(B)] \tag{2}$$

where $|B|$ denotes the number of players in the set (*coalition*) B. $\nu(B \cup \{j\}) - \nu(B)$ is player j's *marginal contribution* to coalition B, i.e. the increase in the joint payoff for coalition B when player j is added to it.

Equivalently, one can also write the Shapley value by summing over the coalitions *containing* player j and expressing his marginal contribution by *removing* him from the coalition. In fact, this is the way Shapley (1953) expressed ϕ_j in his seminal paper. The alternative expression is

$$\phi_j(v) = \frac{1}{k!} \sum_{B \subseteq S := \{\mathcal{A} | j \in \mathcal{A}\}} (|B| - 1)!(k - |B|)![v(B) - v(B \setminus \{j\})] \tag{3}$$

We can readily see that to calculate Shapley values, we need to sum over $\sum_{|B|=1}^{k} \binom{k-1}{|B|-1} = 2^{k-1} - 1$ different marginal contributions, which can also be derived from the fact that $|\mathcal{P}(\mathcal{A})| = \sum_{|B|=0}^{k} \binom{k}{|B|} = 2^k$ and excluding the empty set since $v(\emptyset) = 0$.

A last, equivalent definition which will be used later is based on permutations of $\mathcal{A}$. Let $\mathbf{O}$ be the set of all permutations of $\mathcal{A}$, such that $|\mathbf{O}| = k!$. Furthermore, let $r_k(j)$ denote the rank of player j in a given permutation $o \in \mathbf{O}$. We can then define $L := \{\mathbf{O} \mid j \in \mathbf{O}, r_k(l) < r_k(j)\}$, the set of all players l preceding player j in this permutation. The alternative expression for the Shapley value is then written

$$\phi_j(v) = \frac{1}{k!} \sum_{o \in \mathbf{O}} [(v(L \cup \{j\}) - v(L)] \tag{4}$$

By the weighted averaging over all possible coalitions of players, the Shapley value takes into account all possible interactions between them. The marginal contributions are weighted such that each subset size receives the same importance in determining the payoff allocation. The Shapley value can not only be used to allocate (positive) payoffs; it can equivalently applied to divide up *costs* between players. Moreover, even when sharing payoffs, an individual payoff ϕ_j can be negative when player j has a negative average marginal contribution. This may be less common for applications in game theory (as other members of a coalition would not be likely to accept a player who decreases the joint payoff), but relevant for applications in explainable machine learning, as will be implied in the next section.

2.2 The Shapley Value in Machine Learning

By considering all possible interactions of players and uniquely fulfilling (A1)-(A4), the concept of the Shapley value has recently attracted interest from the domain of machine learning. A similar idea first appears in Kruskal (1987), who proposed to measure the importance of predictors in linear regression models by considering all possible permutations of them: For a given ordering of the predictors, the remaining unexplained variance in the target variable accounted for by predictor $x_j, j \in \{1, ...k\}$ is calculated. To obtain a relative importance measure, this is done over all $k!$ possible orderings and then averaged for each predictor. Also in the context of linear regression models, Lipovetsky & Conklin (2001) first apply the Shapley value for assigning predictor importances. Using the coefficient of determination (i.e., the multiple R^2) as the joint payoff (assigned by the characteristic function v) of the predictors x_j, Lipovetsky and Conklin formulate a cooperative game between the predictors to distribute "fair" shares of the R^2. In this study, the Shapley value approach showed more reasonable point estimates of relative importance for the real-world dataset used

and considerably lower variance over bootstrap resamples than existing methods to measure predictor importance. In general, one can think of many ways to define ν, for example via goodness of fit measures, other evaluation metrics, or, most commonly, on the scale of the model predictions. Otherwise, most common approaches to model explainability based on the Shapley value share some common characteristics: The predictors take on the role of players in a cooperative game $G_i, i = \{1, .., n\}$, one for each observation. Then for each i, a share of ν is assigned to each predictor, which can be averaged over all observations to obtain a global effect or importance measure of any predictor. In the following, the focus will be on the case where ν is directly related to the model's predictions. This will lead to intuitive Shapley values which can explain the predictions of any complex supervised predictive model in an additive way and on the scale of the target variable, independent of the predictor scales.

The general idea will be the following: Given a predictive model and for any given observation i each predictor can be assigned a "fair" share of its contribution to the model's prediction $\hat{y}_i$. To do so, the predictors $x_j, j \in \{1, ...k\}$ are seen as players in cooperative games $G_i(\mathcal{X}, \nu_i)$ where $\mathcal{X} = \{x_1, ..., x_k\}$ denotes the set of all predictors. The characteristic function $\nu : \mathcal{P}(\mathcal{X}) \mapsto \mathbb{R}$ is related to the model predictions. The axioms uniquely fulfilled by the Shapley value express desirable properties for this assignment of "fair" shares. Translated into the domain of explaining model predictions, the four axioms presented in the previous sections can be expressed as follows:

(A1') Efficiency: Each model prediction should be explained as a sum of attributions to the predictors; there is no unexplained component.

(A2') Symmetry: If two predictors have the same marginal effect on the prediction in all possible predictor subsets, they should be assigned equal contributions.

(A3') Dummy player: If a predictor has no marginal effect on the prediction in all possible predictor subsets, no contribution should be assigned to it. Most notably, this means that predictors not contained in a model are assigned a zero share of its predictions.

(A4')Additivity & Linearity: Many common ensemble models create predictions by averaging over the predictions of multiple base models. Predictors should receive the same importance no matter whether it was directly calculated or obtained by averaging over the base models. This can be relevant in cases like stacking ensembles (Breiman (1996)).

Štrumbelj & Kononenko (2010) first applied the game-theoretic concept of the Shapley value to explain predictions from any classification model. For a given observation i, the predictors contained in the model are treated as players in a cooperative game. First, a class level of the target variable is chosen. The characteristic function is defined similarly as before $\nu : \mathcal{P}(\mathcal{X}) \mapsto [0, 1]$ with $\nu(\{\emptyset\}) = 0$. The collective payoff here is tied to the model prediction for the chosen target variable class. To insure that $\nu(\{\emptyset\}) = 0$, ν cannot represent the model's predictions directly (since the prediction of a null model is not 0 in general). Rather, ν assigns to each possible combination of the predictors contained in a predictive model the difference between the actual prediction and the prediction of the null model. Let Y denote the target variable and $f : \mathcal{X} \mapsto Y$ describe the true data generating process. With a model $\hat{f} : \boldsymbol{x} \mapsto \boldsymbol{y}_{\{n \times 1\}}$

approximating f using the design matrix $\mathbf{x}$, let $\hat{\mathbf{y}}_i(C) := \hat{f}(\{\mathcal{X} : x_j \in C \subseteq \mathcal{X}\})$ be the vector of predicted values for the units $i \in \{1, ..., n\}$ with predictor set C in the model. Then, we can define for any i the characteristic function of this predictor set as

$$\nu_i(C) = \hat{y}_i(C) - \hat{y}(\emptyset)$$

For the predicted value of the null model $\hat{y}_i(\emptyset)$, one could plug in $\overline{y} := \frac{1}{n}\sum_{i=1}^{n} y_i$, as this is the "best guess" one can make about the target variable when no predictors are available. Another justification is that predictive problems are often stated as modeling $\mathbb{E}(Y_i \mid \mathbf{X}_i)$, where $\mathbf{X}_i$ denotes the vector of predictors for observation i. Without any predictors, the conditional expectation becomes the unconditional one, which can be consistently and unbiasedly estimated by $\overline{y}$. For some models (like GLMs), we have $\hat{y}_i(C) = \overline{y}, \forall C \subseteq \mathcal{X}$. However, models like tree-based ensembles and support vector classifiers may have good abilities in discriminating between classes, but are known to be badly calibrated (Niculescu-Mizil & Caruana, 2005), such that the above equality is not guaranteed. As such algorithms can usually not fit null models and due to the calibration issue, assuming $\hat{y}(\emptyset) = \overline{y}$ might not be the best choice. Alternatively, one may choose to plug in the average prediction of the full model $\overline{\hat{y}}(\mathcal{X})$ instead.

Note that the marginal contributions $\nu(C) - \nu(C \setminus \{j\})$ reduce to the prediction difference $\hat{y}_i(C) - \hat{y}_i(C \setminus \{j\})$. Thus, given a predictive model $\hat{f}$, the Shapley value for a given predictor j at observation i can be expressed as

$$\phi_{ij}(\hat{f}(\mathbf{X}_i)) = \frac{1}{k!} \sum_{C \subseteq S := \{\mathcal{X} : x_j \in \mathcal{X}\}} (|C| - 1)!(k - |C|)![\hat{y}_i(C) - \hat{y}_i(C \setminus \{j\})] \tag{5}$$

equivalently to equation (3) with $\hat{y}(\emptyset)$ as one of the estimators discussed above. It also follows that $\nu_i(C) = \hat{y}_i(C)$ in this setting.

The notation $\hat{f}$ emphasizes Shapley values as a *post-hoc* explanation method. In the general case, they are only obtained *after* fitting a model $\hat{f}$ whose predictions ought to be explained. Finally, the ϕ_{ij} can then also be averaged over the observations to obtain a measure of global predictor importance and an "effect" by how much a given predictor tends to change the predictions on average.

This game formulation with ν representing prediction differences from an average or uninformative (null model) prediction can also be motivated by the fact that often, explanations are sought for predictions that are deemed "abnormal" or "unusual". In many applications, predictions in the extremes have the strongest implications in some decision being taken based on a machine learning model (for instance, a bank refusing a loan when the probability of default is estimated high). In such cases, it may make sense to let the characteristic function ν take on a relatively large value to minimize the effects of variability and uncertainty in the data on the explanation. In general, the characteristic functions considered for applications in explainable machine learning may look very different from those common in game theory. For instance, there will be many negative "payoffs" in the game formulation above. The use of the Shapley value here is motivated mainly by the unique solution of how to allocate shares

of predictions to predictors, fulfilling the set of axioms given before. In general, there is a lot of flexibility in the choice of ν (see, for instance, Lipovetsky & Conklin (2001)).

Some flexibility is also given in how to define a player in the cooperative game. For instance, categorical predictors are usually treated as one variable although they are often decomposed into multiple dummy variables before model fitting. A similar case can arise when including transformations (e.g. polynomials) of predictors alongside their untransformed values, as is commonly done in regression modeling. Due to their direct dependence, these variables should also be regarded as one player. Another possibility is to regard domain-specific criteria in the area of application and group together multiple variables with a similar domain-specific meaning (Jullum et al., 2021), which alleviates the computational burden of calculating Shapley values and eases interpretation when many predictors are present.

As implied in the previous section, calculating exact Shapley values involves $2^k - 1$ predictor sets to be evaluated. This is not computationally feasible except when there are only a relatively small number of predictors. The literature has thus focused on the task of finding computationally feasible approximations, which are the topic of the following chapter.

3. Approximating Shapley Values

In the context of complex statistical and machine learning models, the characteristic function ν is generally not available in closed form. Hence, to determine all marginal predictor contributions for calculating Shapley values requires fitting a given model with k predictors for all possible combinations of them, yielding $2^k - 1$ different models to estimate (when excluding the null model). Hence, exact calculation of Shapley values has exponential time complexity, making such calculations infeasible when using somewhat complex models with even a moderate number of predictors. For instance, assume it takes (on average over all elements of $\mathcal{P}(\mathcal{X})$) one second to refit the model at hand. Thus on one day, $60^2 \cdot 24 = 86400$ models could be run. This corresponds to a predictor set with cardinality $\lfloor \log_2(86400) \rfloor = 16$, which is a relatively low number in current machine learning practice. In the same setting, fitting all possible models with 50 predictors would take more than 35 million years.

Facing the same issue in game theory, Mann & Shapley (1960) first applied a Monte Carlo estimator to the estimation of Shapley values. Drawing from the idea of evaluating only sampled coalitions, Štrumbelj & Kononenko (2010) proposed an approximation based on sampling with replacement pairs of predictor permutations (the set $\mathbf{O}$ in equation (4)). Additionally, to avoid refitting the model multiple times, the authors implemented another simplification: The full model is fitted once and models for all other subsets of predictors are simulated by predicting with the full model on data where predictor values are shuffled between observations, mimicking absence of predictors not in the subset of interest. In some applications, there may also exist predictors which naturally have uninformative values; these can then also be plugged in to simulate absence of predictors. These approaches however ignore any dependencies between predictors, thus relying on the assumption that predictors are independent.

Lundberg & Lee (2017) develop a general framework to include various explanation methods such as Shapley values and a certain specification of LIME into a class of additive explanation methods which share the common characteristic of decomposing a model's predictions into a sum of predictor attributions $\phi_j, j \in \{1, ..., k\}$, such that

$$\hat{y}_i(C) = \phi_o + \sum_{j:x_j \in C \subseteq \mathcal{X}} \phi_{ij}$$

Here again, $C \subseteq \mathcal{X}$ denotes a given set of predictors. The authors call their framework *SHAP* ("*SH*apley *A*dditive ex*P*lanations"). To recover Shapley values from the above definition (i.e. to insure $\nu(\emptyset) = 0$), we can set ϕ_0 to $\hat{y}(\emptyset)$ as discussed before. Based on the above definition, the following formulation of the characteristic function is then given, allowing for estimation of Shapley values by only fitting the full model once. Assume the predictors X_j are random variables. The characteristic function is stated (for any observation i) as the expectation of the prediction on the full model given only observations of predictors in set C:

$$\nu_i(C) = \mathbb{E}[\hat{y}_i(\mathcal{X}) \mid X_{ij} = x_{ij}, \forall X_j \in C \subseteq \mathcal{X}] \tag{6}$$

This formulation has the advantage that only the full model has to be estimated. For any predictor subset C, the other predictors contained in $\mathcal{X} \setminus C$ are set to values that are

"uninformative", such that they have no influence on the model prediction (similarly to Štrumbelj & Kononenko (2010)). Given these newly generated data points and the actual observations of $X_j \in C$, it is then sufficient to predict with the full model to obtain the marginal contributions and thereby the Shapley value approximations. The major challenge will be to generate data for $\mathcal{X} \setminus C$ such that these predictors have no influence on the model output.

While only fitting the full model and then predicting with it will save computational resources, the exponential time complexity of going through all possible predictor subsets remains. Like Štrumbelj & Kononenko (2010), Lundberg & Lee (2017) thus follow a two-part strategy by first sampling predictor combinations to reduce the number of marginal contributions to evaluate. Most importantly however, they apply an important result from the game theory literature (Charnes et al., 1988) to obtain Shapley values for all predictors at once by solving a weighted least squares problem, which further speeds up computation. Lundberg and Lee call this approach *KernelSHAP*, which will be treated in what follows. Based on the SHAP framework in general, other approaches to approximating Shapley values in the machine learning setting have been developed. For instance, Lundberg et al. (2020) propose a method specific to tree-based models which makes use of the model structure for faster computation. Jethani et al. (2022) propose a fast approximation for settings with large k based on a surrogate model. Since the focus of this work is on model-agnostic methods for models in tractable dimensions, we limit ourselves to treating topics related to the now widely used KernelSHAP approach.

3.1 Kernel SHAP

Charnes et al. (1988) showed that for a class of solution concepts for cooperative games, the solutions (that is, allocations of joint payoffs) can also be obtained by solving an appropriately formulated weighted least squares problem. In the case of Shapley values, the vector $\boldsymbol{\phi} := (\phi_1 .... \phi_k)^T$ is recovered by solving the objective

$$\min_{\boldsymbol{\phi}} \sum_{A \subseteq \mathcal{A}} \binom{k}{|A|}^{-1} \frac{k-1}{|A|(k-|A|)} (v(A) - \sum_{j \in A} \phi_j)^2$$

under $\sum_{j \in \mathcal{A}} \phi_j = v(\mathcal{A})$, the efficiency constraint from (A1).

Adopted by Lundberg & Lee (2017) for the setting discussed in the previous section, Shapley values for any observational unit i are obtained as

$$\hat{\boldsymbol{\phi}}_i = \underset{\boldsymbol{\phi}_i}{\operatorname{argmin}} \sum_{C \subseteq \mathcal{X}} \binom{k}{|C|}^{-1} \frac{k-1}{|C|(k-|C|)} (\hat{v}_i(C) - \mathbb{E}[\hat{y}(\emptyset)] - \sum_{j: X_j \in C} \phi_{ij})^2 \tag{7}$$

subject to $\mathbb{E}[\hat{y}(\emptyset)] + \sum_{j: X_j \in \mathcal{X}} \phi_{ij} = v(\mathcal{X})$ for all i.

The advantage over the definition in (5) is that for each i, the estimation of the Shapley values can be conducted for all predictors simultaneously, using a closed-form solution instead of going through each X_j seperately and calculating each marginal contribution. However, the exponential time complexity remains: One still needs to go through all possible predictor subsets to calculate an estimate for $v_i(C)$ as defined in (6). To alleviate this issue, one can

sample with replacement a subset of all possible predictor combinations $S \in (\mathcal{P}(\mathcal{X}) \setminus \{\emptyset, \mathcal{X}\})$ and solve the objective using only this subset, yielding a faster approximation. Since there is a huge disparity between the weights depending on the subset size $|C|$, the variance of the estimator can be reduced by a refined sampling scheme: Instead of drawing S from a discrete uniform distribution, $\hat{\boldsymbol{\phi}}_j$ is obtained by drawing S from a distribution proportional to the weights

$$\binom{k}{|C|}^{-1} \frac{k-1}{|C|(k-|C|)}$$

and then removing the weights from the objective function (7). It remains the question how to estimate $v_j(S)$ for any $S \in (\mathcal{P}(\mathcal{X}) \setminus \{\emptyset, \mathcal{X}\})$. The characteristic function for the full and empty set need not be evaluated, as they are known or assumed to be known, respectively.

Slightly abusing notation, let $p()$ denote a (joint) probability density or mass function. We further denote $\mathbf{X}_C$ as the predictors $C \subseteq \mathcal{X}$ with realizations $\mathbf{x}_C$. Without loss of generality, let C contain only continuous predictors. We can then write the characteristic function in (6) for any $j \in \{1, ..., n\}$ as

$$v_j(C) = \int_{-\infty}^{\infty} \hat{y}_j(\mathbf{X}_{\mathcal{X}\setminus C}, \mathbf{x}_C) p(\mathbf{X}_{\mathcal{X}\setminus C} \mid \mathbf{X}_C = \mathbf{x}_C) \, d\mathbf{x}_{\mathcal{X}\setminus C}$$

If all $X_j \in \mathcal{X}$ are mutually independent, the conditional distribution is equivalent to the marginal one and the characteristic function becomes

$$v_j(C) = \int_{-\infty}^{\infty} \hat{y}_j(\mathbf{X}_{\mathcal{X}\setminus C}, \mathbf{x}_C) p(\mathbf{X}_{\mathcal{X}\setminus C}) \, d\mathbf{x}_{\mathcal{X}\setminus C} \tag{8}$$

for all $C \subseteq \mathcal{X}$.

Lundberg & Lee (2017) use this independence assumption for constructing a simple Monte Carlo estimator of the integral (8) to estimate the characteristic function for any $C \in \mathcal{X}$: By sampling M times with replacement from the empirical marginal distributions $p(\mathbf{x}_{\mathcal{X}\setminus C})$ (usually from the training data), the expectation in (5) can be consistently and unbiasedly estimated by the KernelSHAP estimator

$$\hat{v}_j^{\mathrm{KernelSHAP}}(C) = \frac{1}{M} \sum_{m=1}^{M} \hat{y}_j(\mathbf{x}_{\mathcal{X}\setminus C}^{(m)}, \mathbf{x}_C) \tag{9}$$

where $\mathbf{x}_{\mathcal{X}\setminus C}^{(m)}$ denotes the random draws from the empirical marginal distributions $p(\mathbf{x}_{\mathcal{X}\setminus C})$. The KernelSHAP estimator for $\boldsymbol{\phi}_j$ is finally obtained by estimating $\hat{v}_j^{\mathrm{KernelSHAP}}(S)$ for all subsets S that were sampled in the previous step and plugging in this expression for $\hat{v}_j(C)$ in equation (7).

For most applications, the assumption of mutual independence between the predictors is severely violated. As a consequence, the joint empirical distribution of predictors resulting from sampling only the marginals of $\mathbf{X}_{\mathcal{X}\setminus C}$ will cover unrealistic regions of the predictor space that a model has never encountered in training. This might lead to predictions $\hat{y}_j(\mathbf{x}_{\mathcal{X}\setminus C}^{(m)}, \mathbf{x}_C)$

very far from what would be the actual prediction $\hat{y}_i(C)$ when the model is only trained on the subset C and thereby affect the estimated Shapley values. This problem recently motivated a branch of research aiming to incorporate dependencies between predictors, which is the focus for the rest of this work.

3.1.1 Accounting for Dependent Predictors

Aas, Jullum, et al. (2021) develop several methods to generate more realistic data points for estimating the conditional expectation in (6). Their general idea is to estimate the conditional distributions $p(\mathbf{X}_{\mathcal{X}\setminus C} \mid \mathbf{X}_C = \mathbf{x}_C)$ directly instead of relying on the assumption of independence. To this end, the authors propose four different methods designed for a setting where all predictors are continuous:

(M1) Multivariate Gaussian: Let $\mathcal{N}(\cdot)$ denote the multivariate normal distribution. Assuming that $\mathbf{X}_{\mathcal{X}} \sim \mathcal{N}(\mu, \Sigma)$ with expectation vector $\mu = (\mu_{\mathbf{X}_C}, \mu_{\mathbf{X}_{\mathcal{X}\setminus C}})$ and covariance matrix

$$\Sigma = \begin{bmatrix} \Sigma_{\mathbf{X}_C\mathbf{X}_C} & \Sigma_{\mathbf{X}_C\mathbf{X}_{\mathcal{X}\setminus C}} \\ \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}\mathbf{X}_C} & \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}\mathbf{X}_{\mathcal{X}\setminus C}} \end{bmatrix}$$

for any $C \subset \mathcal{X}$, it follows that the conditional distribution $p(\mathbf{X}_{\mathcal{X}\setminus C} \mid \mathbf{X}_C = \mathbf{x}_C)$ is given by $\mathcal{N}(\mu_{\mathbf{X}_{\mathcal{X}\setminus C}|\mathbf{x}_c}, \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}|\mathbf{x}_c})$ with

$$\mu_{\mathbf{X}_{\mathcal{X}\setminus C}|\mathbf{x}_c} = \mu_{\mathbf{X}_{\mathcal{X}\setminus C}} + \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}\mathbf{X}_C}\Sigma^{-1}_{\mathbf{X}_C\mathbf{X}_C}(\mathbf{x}_C - \mu_{\mathbf{X}_C})$$

and

$$\Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}|\mathbf{x}_c} = \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}\mathbf{X}_{\mathcal{X}\setminus C}} - \Sigma_{\mathbf{X}_{\mathcal{X}\setminus C}\mathbf{X}_C}\Sigma^{-1}_{\mathbf{X}_C\mathbf{X}_C}\Sigma_{\mathbf{X}_C\mathbf{X}_{\mathcal{X}\setminus C}}$$

After estimating μ and Σ by the sample mean and covariance matrix (respectively), one can generate data from the conditional distribution using the above decompositions. The advantage of this method is its analytical tractability; estimation and generating draws from this distribution are straightforward. In practice however, the multivariate Gaussian assumption is quite specific in its distributional assumptions on $\mathbf{X}_{\mathcal{X}}$. This motivates the next method, which requires that only the marginals follow a normal distribution, but the dependencies between predictors can be modeled more flexibly.

(M2) Gaussian copula: Copulae are joint distribution functions with uniformly distributed marginals on $[0, 1]$. They have the advantage that the marginal distributions can be seperated from the dependence structure of the predictors. According to Sklar's theorem (Sklar, 1959), there exists for any k- dimensional cumulative distribution function $H(\cdot)$ with the corresponding marginals $F_1, ..., F_k$ a copula K such that

$$H(X_1, ..., X_k) = K(F_1(X_1), ..., F_k(X_k))$$

for all $\mathbf{X} := (X_1, ..., X_k) \in \mathbb{R} \cup \{-\infty, \infty\}$. When the $F_1, ...F_k$ are continuous, we have

$$H(F_1^{-1}(U_1), ..., F_k^{-1}(U_k)) = K(U_1, ..., U_k)$$

for $\mathbf{u} := (u_1, ..., u_k) \in [0,1]^k$. Plugging in a multivariate normal cumulative distribution function (cdf) for $H(\cdot)$ and the univariate normal cumulative distribution function for all $F_q, q \in \{1, .., k\}$ yields a Gaussian copula.

While this approach requires only the marginals to follow a Gaussian law, it is still restrictive in the type of dependencies it can incorporate. Other copulae could also be employed, but again the Gaussian case has the advantage of analytical tractability and thus a relatively straightforward process for generating data: First, the empirical marginal distributions of all predictors x_j are transformed to follow a Gaussian distribution by an inverse cdf transform of their empirical marginal cumulative distribution functions. Under the assumption that the joint distribution of all transformed X_j is also Gaussian (this limits the dependency types accounted for by this method), samples from the conditional distribution $p(\mathbf{X}_{\mathcal{X}\setminus C} \mid \mathbf{X}_C = \mathbf{x}_C)$ are generated according to (M1) by replacing the X_j by their transformed counterparts. In the end, the X_j are converted back to their original marginal distributions, reversing the first step.

(M3) Empirical conditional distribution: To be more flexible in the type of dependencies and forms of marginal distributions, the authors also develop a nonparametric alternative. Since the classical Kernel density estimators do not provide a method of generating samples from the estimated distribution, they develop a different approach. Using a weighting scheme (involving the Mahalanobis distance) according to the similarity of training observations, the conditional distributions are approximated by taking only a certain proportion of the closest observations into account. Still, bandwidths for different subsets $C \subseteq \mathcal{X}$ have to be chosen. This is done based on the AICc, a corrected version of Akaike information criterion. Along with the weighting scheme, this makes the empirical conditional approach computationally intensive. In their simulation studies, Aas, Jullum, et al. (2021) only employ this method for settings with three predictors. As an alternative, the study proposes to set a bandwidth according to the subset size $|C|$. As another option, they even experimented with a fixed bandwidth for all predictor subsets. As it is not believed that these methods scale well when dimensions of the predictor space increase and due to computational constraints, the nonparametric approach will not be considered when applying the discussed methods (M1)-(M5) in the following chapter.

(M4) Combined approaches: Since non-parametric estimators quickly deteriorate in higher dimensions and due to their aforementioned computational costs, the authors propose to combine the nonparametric approach with the other two. To reap the benefits of nonparametric flexibility and parametric robustness, they suggest to generate data using the empirical conditional approach when conditioning on few predictors (that is, when $|C|$ is small) and use (M1) or (M2) for the other predictor subsets.

In their paper, the authors conducted several simulation experiments to test their newly proposed methods (M1)-(M4). Using settings with 3 and 10 predictors, normal, heavy-tailed and even bimodal predictor distributions with varying degrees of dependency, as well as linear and piecewise constant sampling models, they find their methods clearly outperforming the classical KernelSHAP estimator by Lundberg & Lee (2017) except when predictors are nearly independent. The baseline for the "true" Shapley values was obtained by using numerical integration in the setting with three predictors and Monte Carlo integration sampling directly

from the known population-level conditional distributions otherwise. In the experiments, which of the newly proposed methods worked best depends on the joint of distribution of the predictors. For instance, (M1) incurred the smallest approximation error when the joint predictor distribution was a multivariate Gaussian and (M3) worked best in the experiments involving bimodal distributions.

A big drawback of all approaches just discussed is that they are designed for continuous predictors. In practice, there is usually a mix of categorical, count and continuous explanatory variables present in predictive models. Motivated by this, Redelmeier et al. (2020) develop a procedure based on *conditional inference trees* (Hothorn et al., 2006), which can be applied to models with all sorts of predictors.

(M5) Conditional inference trees: Like other tree-based models, conditional inference trees (*ctrees*) divide the predictor space recursively by applying binary splits until some stopping criterion is fulfilled. As opposed to other tree-based models, conditional inference trees select the next predictor to split by conducting a statistical test on the dependence between all predictors and the target variable. When the global null hypothesis of independence cannot be rejected, the tree is not split any further. Otherwise, the predictor with the lowest p-value from the corresponding local test is selected as the next splitting variable. After selecting a split predictor, the splitting point can be chosen according to another criterion. The tree depth is controlled by the choice of a significance level α, where decreasing α will also decrease the number of tree splits since the global tests tend to fail to reject more often. Another difference to conventional tree-based models is that the ctree algorithm can also handle multidimensional target variables. This property is exploited by Redelmeier et al. (2020) to apply this framework for estimating the conditional distributions $p(\mathbf{X}_{\mathcal{X}\setminus C} \mid \mathbf{X}_C = \mathbf{x}_C)$. To do so, conditional inference trees are trained with $\mathbf{X}_{\mathcal{X}\setminus C}$ as the predictor set and $\mathbf{X}_C$ being the target variable. In the following, the end nodes for $\mathbf{X}_C = \mathbf{x}_C$ are found in the fitted tree, from which we sample with replacement values from $\mathbf{X}_{\mathcal{X}\setminus C}$ to finally plug into the KernelSHAP estimator (9). When conducting simulation studies, Redelmeier et al. found their approach outperforming the original KernelSHAP approach, especially in higher dimensions and degrees of predictor dependence. Moreover, they showed that converting categorical predictors into sets of dummy variables to apply (M1) induced much higher running times compared to the conditional inference tree approach.

This approach, along with the others presented in this section, is implemented in the `R` (R Core Team, 2023) programming language by the package `shapr` (Sellereite et al., 2024). To be able to account for discrete predictors and due to the accessible implementation along with the other methods, all these approaches are tested on three diverse, real-world datasets in the coming chapter.

4. Applications to Real Datasets

Assessing the accuracy of approximations to Shapley values suffers from some major problems: Firstly, one needs the "true" (population-level) Shapley values for comparison. This requires knowledge of the characteristic function v, which is generally not available in closed form except for some simple cases such as linear models. As is apparent from the simulation studies of Aas, Jullum, et al. (2021) and Redelmeier et al. (2020), establishing the true population Shapley values even for those kinds of models can be computationally intensive. In fact, the ability to establish such a population baseline may depend on the game formulation. Moreover, the use of Shapley values in a machine learning context is motivated by the use of complex models which are employed precisely because they are believed to approximate the (complex) underlying data generating process more accurately. Regardless of the actual data generating process (the "true model"), Shapley values as a post-hoc explanation method always depend on the data generating process indirectly, that is through some predictive model. Hence, there seems little practical value in establishing a population baseline. Rather, to compare different approximation methods, the exact Shapley values obtained through fitting all $2^k - 1$ possible models (excluding the null model) on real-world data are taken as reference to compare the different approximations. To keep the setting application-oriented, a model is first fitted on training data before explanations are generated on the test data not yet seen by the model. This has the slight disadvantage that for non-calibrated models such as tree ensembles, even the exactly calculated Shapley values are not guaranteed to fulfill (A1'). As discussed before, (M1), (M2), (M5) and the original KernelSHAP algorithm of Lundberg & Lee (2017) will be compared against the baseline of exactly calculated Shapley values according to (5).

To cover a range of problems, different predictor distributions, and sample sizes, three datasets were picked from the UCI Machine Learning Repository (Kelly et al., 2023), which offers a wide range of datasets from different domains, according to the following criteria:

- The number of predictors in the final model does not exceed 15, such that computation of exact Shapley values is feasible on a regular computer. The number of predictors should not be so small to render the experiment irrelevant for practical purposes, so a minimum number of 5 predictors is chosen. An additional advantage when defining a narrow range for the number of predictors is that one can reasonably tune model hyperparameters over the same grid (the optimization of some model hyperparameters strongly depends on the number of predictors).

- At least one of the datasets contains both numerical and categorical predictors, as in the vast majority of real-world application scenarios.

- The datasets do not contain missing values or many redundant variables to keep preprocessing requirements minimal.

- The datasets have a moderate sample size to keep the computational burden for modeling and calculation of exact Shapley values at a reasonable level.

- At least one dataset contains a target variable suitable for a classification problem and at least one for a regression task, such that the most important general problems are covered.

- There is sufficient variation between the datasets with respect to the joint distributions of predictors and target variable (and thereby their dependence structures). This insures that a range of real-world scenarios is covered.

- The datasets come from different domains of application and are not synthetic. However, the selection is mainly data driven and not motivated by the domains of application themselves.

- Ideally, the dataset should be well-known, for example through being used in a widely known paper, such that any issues with the data are common knowledge.

Based on these criteria, three datasets were chosen: *Rice*[2] (Cinar & Koklu, 2019) and *Seismic bumps*[3] (Sikora & Wróbel, 2010) for classification and *Concrete*[4] (Yeh, 1998) for a regression problem. They do not contain missing values. The most important attributes of the datasets are given in Table 1, including the total number of predictors k and how many were finally used in modeling (k_{final}), along with information on the types of predictors (categorical, count or continuous). Some further description and exploratory analysis is contained in the following subsection.

TABLE 1: Attributes of datasets

Dataset	n	k	k_final	Continuous	Count	Categorical	Target
Rice	3810	7	6	7	0	0	binary
Seismic bumps	2584	18	13	6	8	4*	binary
Concrete	1030	8	8	8	0	0	positive real

*the maximum number of categories is 4

4.1 Data Descriptions

The rice dataset contains measures of two sorts of rice grains. The goal is to predict the sort of rice (*Cammeo* or *Osmancik*) with these measures, which were extracted by processing pictures of rice grains taken by a computer vision system. For details, refer to Cinar & Koklu (2019), who used the data for training various classification models which can potentially used for food processing. The data has only a few predictors, which are all continuous with most having marginal distributions close to the Gaussian (see Fig. 1). It can also be seen that most predictors already have decent discriminative power by themselves, while mostly being far from perfect interdependency (Fig. 4). Thus, classification models on this dataset are expected to do quite well, as can also be seen in the aforementioned paper. One predictor was excluded from modeling due to an almost perfect monotonic relationship with another.

[2]https://archive.ics.uci.edu/dataset/545/rice+cammeo+and+osmancik (retrieved: Dec 03, 2023)
[3]https://archive.ics.uci.edu/datasets?search=seismic-bumps (retrieved: Dec 03, 2023)
[4]https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength (retrieved: Dec 03, 2023)

A second classification task is given by the *Seismic bumps* data, where the target variable is given by an indicator whether a *high-energy* seismic bump (>10000 Joule) occured during a given shift in a coalmine. This event is termed the *hazardous state* due to the associated dangers of rockburst. Models predicting such seismic hazards could be used in risk assessment for coal miners. The data were taken in a polish coal mine and contain various measurements of seismic activity taken in the previous shifts as well as information about the shift itself. Again, more details about the data and modeling approaches can be found in Sikora & Wróbel (2010). As can be seen from Figure 2[5], this data is not as well-behaved; it includes heavily skewed and categorical predictors with high imbalance between classes. There does not seem to be good discrimination between classes when only considering one predictor at a time. Moreover, the target variable is also highly imbalanced. Some variables contained in the dataset were removed because of either a perfect dependency to another or due to only taking on one value over all observations. It is believed to be the hardest modeling problem and has the most predictors of all datasets considered.

Lastly, the *Concrete* data provides information on proportions of concrete mixture ingredients, the age of the mixture and the resulting compressive strength, which will be the target variable in a regression problem. According to Yeh (1998), the target is a "*highly non-linear*" function of the predictors. The dataset contains only continuous variables, but as can be seen from Figure 3, these do not necessarily follow well-behaved distributions. All potential predictors in this dataset were used for modeling.

[5]note that categorical variables were converted into integer numbers during the exploratory analysis. Since all categorical predictors are either binary or have a natural ordering, this should not result in a huge loss of information. For modeling purposes, these variables are properly treated as factors. Moreover, the target variables were renamed as "target" for each dataset.

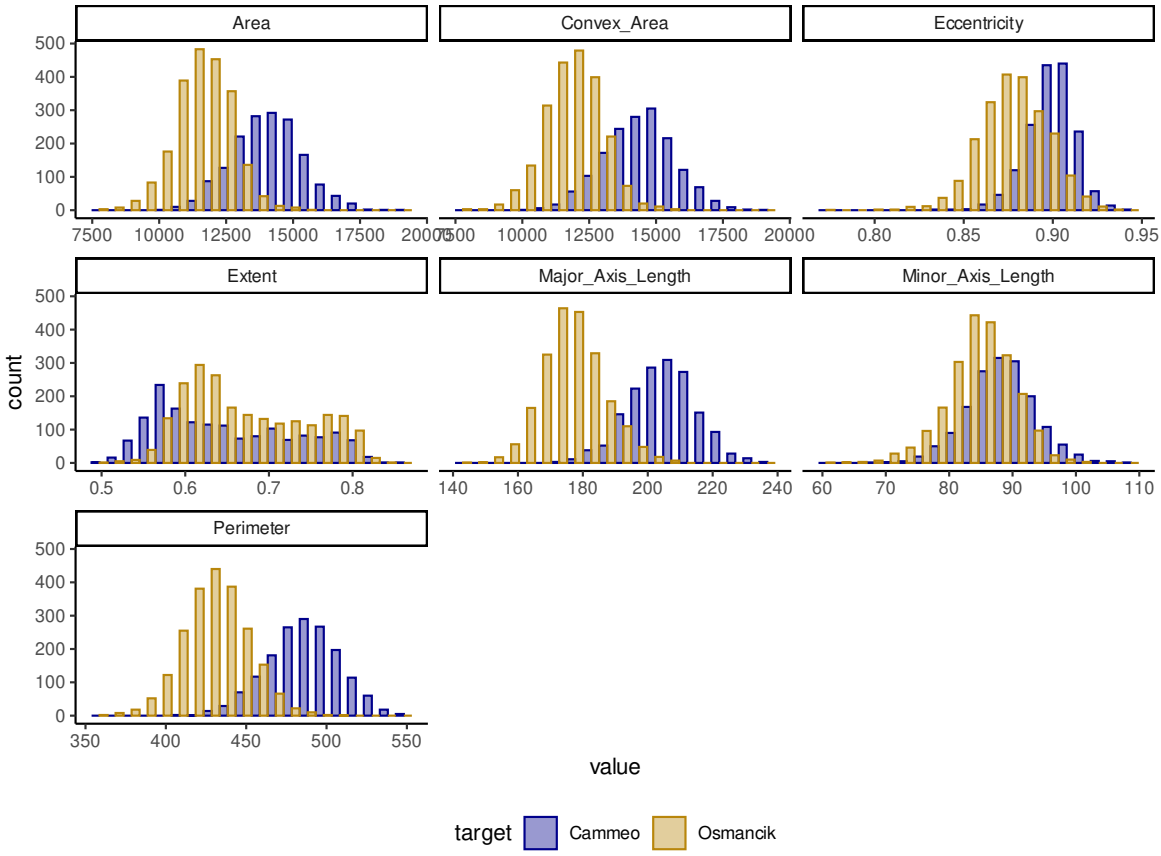

FIGURE 1: Rice data: Histograms of predictors, by response

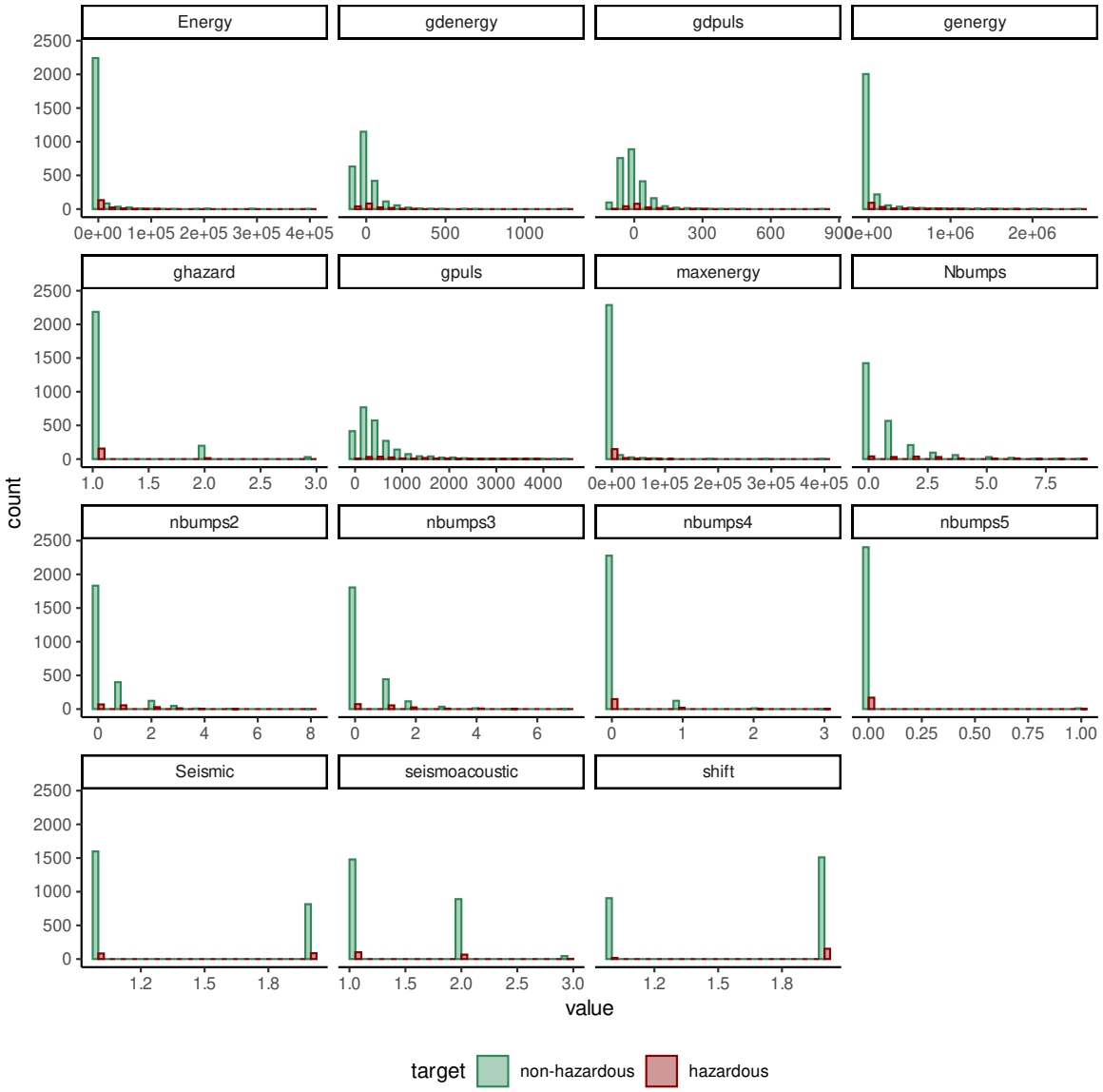

FIGURE 2: Seismic bumps data: Histograms of predictors, by response

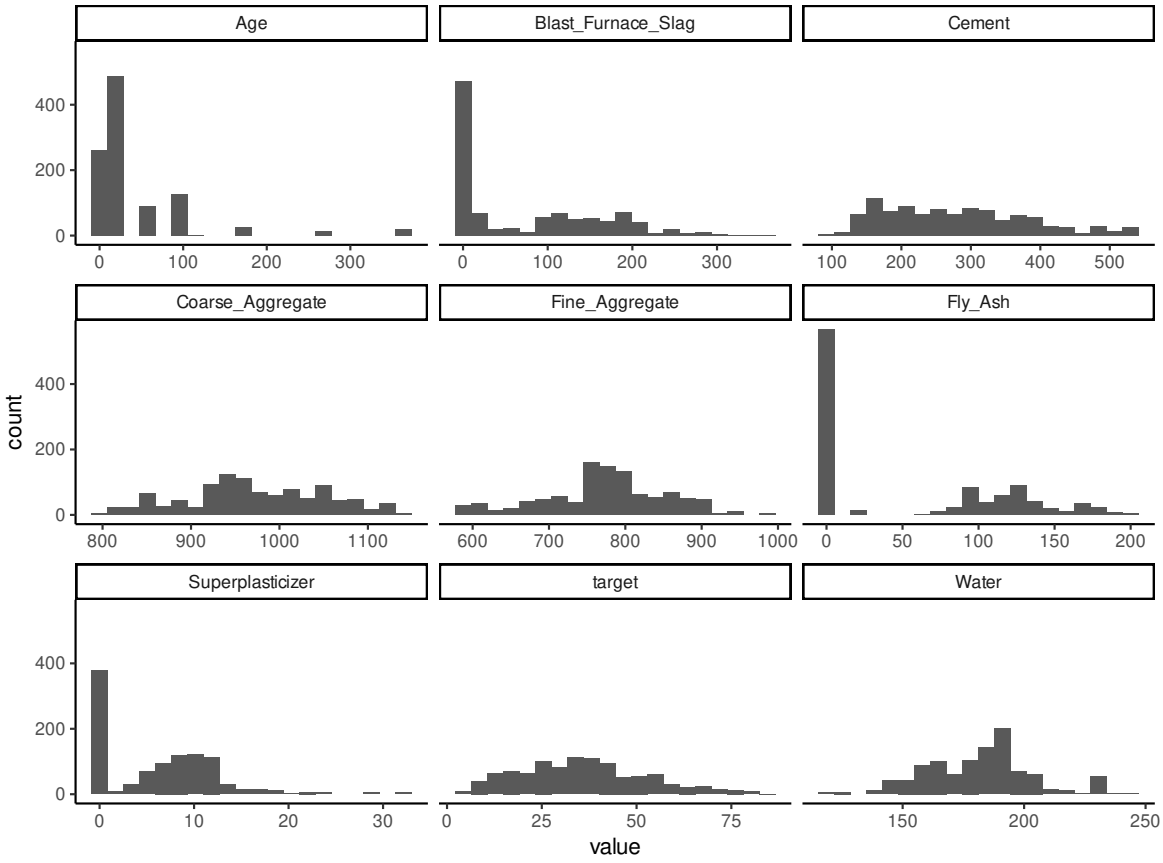

FIGURE 3: Concrete data: Histograms of variables

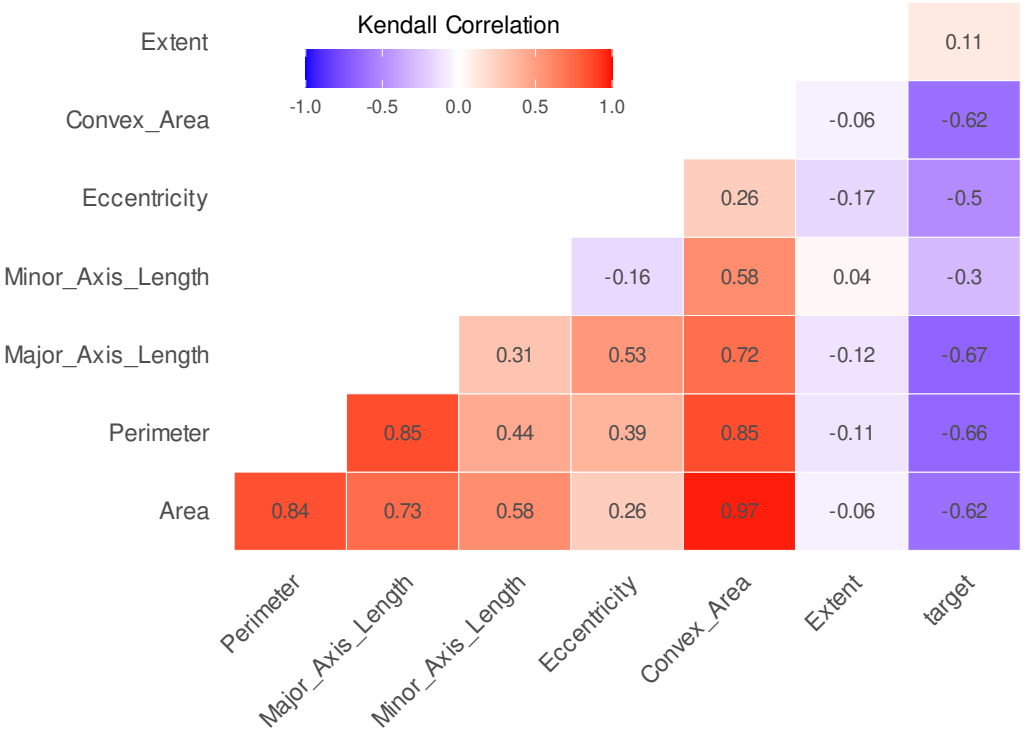

FIGURE 4: Kendall correlation matrix: Rice dataset

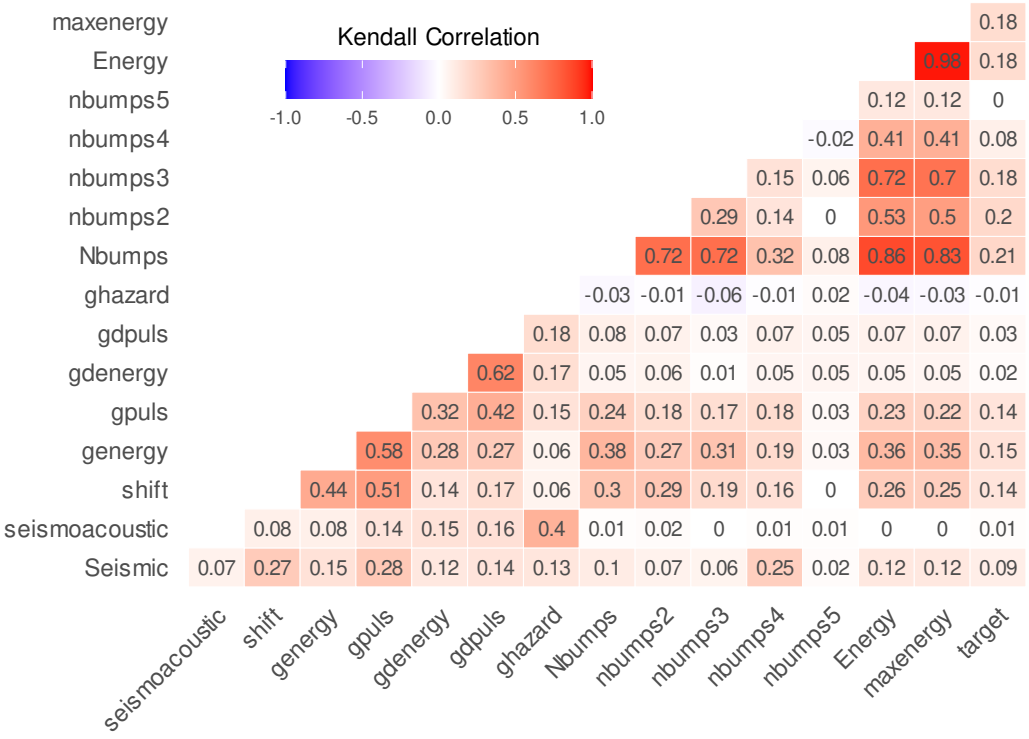

FIGURE 5: Kendall correlation matrix: Seismic bumps dataset

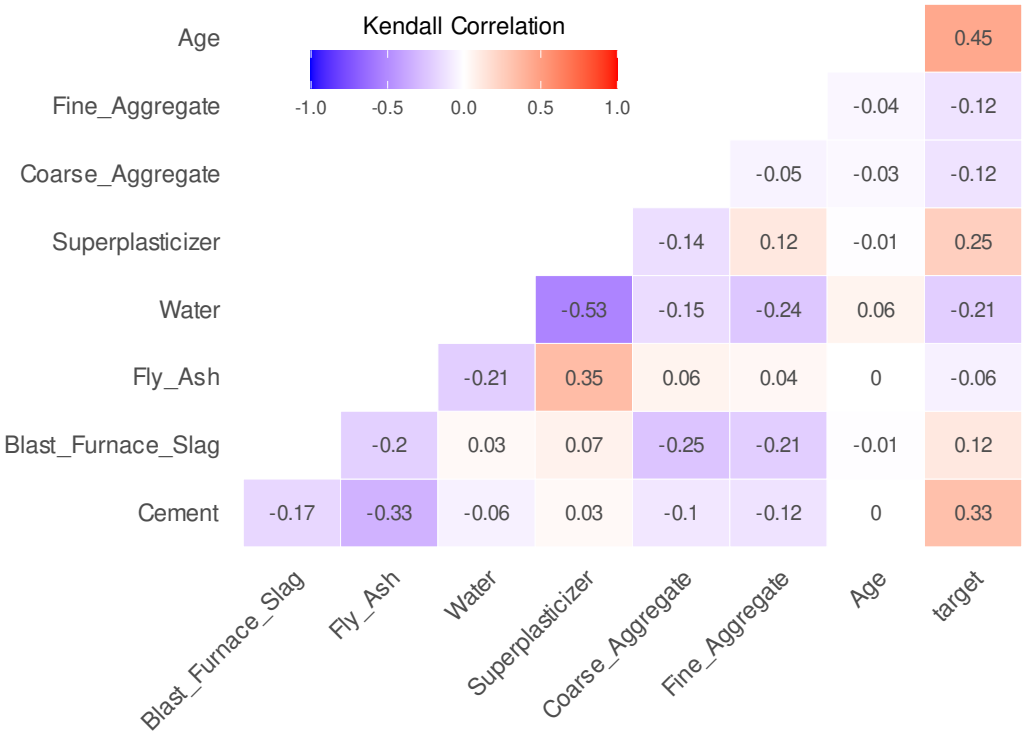

FIGURE 6: Kendall correlation matrix: Concrete dataset

4.2 Model Training

In the following, experiments are conducted with three different model types on all datasets: The aforementioned *GLMs* (Nelder & McCullagh, 1983), *Random Forests* (Breiman, 2001) and *XGBoost* (Chen & Guestrin, 2016). Due to implementation issues with treating categorical variables, XGBoost was not considered for the *seismic bumps* dataset. Regarding the GLMs, logistic regression was chosen for classification and a Gamma GLM with log link for the regression problem to incorporate possible multiplicative effects and keeping the target variable in its admissible range.

All three model types are commonly used in business and scientific applications and are therefore of high practical relevance. Moreover, they represent model types with varying degrees of complexity. The logistic regression model is a classical tool with relatively low computational burden, high interpretability but limited flexibility concerning the functions describing the relationship between predictors and target variable it can approximate. The two tree-based models have lower interpretability and higher computational costs, but can detect more complex structures in the underlying data generating process (i.e. non-linearities and interactions), need very little data preprocessing and known for their high predictive accuracy. Lastly, all of the three models are natively supported in the `shapr` package (Sellereite et al., 2024) for R, which implements the approach suggested by Aas, Jullum, et al. (2021) along with the original KernelSHAP version.

To insure a realistic evaluation of the models and to mimic common use cases where model predictions on new data have to be explained, all datasets are randomly split: 80% will

be used for model training and the remaining 20% for model evaluation and to apply the explainability methods to. 5-fold cross-validation is conducted on the training set in order to avoid overfitting when optimizing hyperparameters. Hyperparameters were tuned across low-dimensional grids around the default values to obtain models with a reasonable performance and thus keeping the setting realistic. For details, refer to Table 2 and the `R` code in the Appendix. The final hyperparameters and the best performing model were selected by choosing the configuration with the minimal logloss for classification or the minimal mean squared error for the regression task. "Minimal" here refers to the lowest average across folds. After finding the best model for each dataset, these models are fitted on the whole training set before final evaluation. The final models are given in Table 3 along with their hyperparameter configurations. In models without hyperparameters, fitting is conducted on the whole training set right away. To assess the quality of the final models, the area under the ROC curve (Bradley, 1997) is calculated for classification problems, where a value of 1 indicates perfect discrimination while a value of 0.5 corresponds to random guessing knowing only the overall class probabilities. This method has the advantage of not requiring a specific probability cutoff (the optimality of which depends on the practical problem at hand) for assessing the overall discriminatory power of a classifier. In the regression case, the mean absolute error (MAE) is taken as the evaluation criterion. The final evaluation metrics are given in table 4.

All calculations are performed in `R`. Model fitting and hyperparameter tuning is conducted with the `tidymodels` package (Kuhn & Wickham, 2020), which offers a unified framework to tune and fit multiple different models at once, keeping the code more maintainable. Table 1 shows a short description of the models being fit, including the hyperparameters tuned and which `R` package was utilized by `tidymodels` for model fitting.

Table 2: Models, hyperparameters and their tuning ranges

Model	Hyperparameter	Range	Package
GLM*	-	-	*glm*
Random Forest	min. number of observations in terminal nodes	[5,14]	*ranger*
	number of predictors sampled for tree splits	[1,10]	
	number of trees in the ensemble	500**	
XGBoost	learning rate	(0, 0.1]	*xgboost*
	min. number of observations in terminal nodes	[4, 20]	
	number of predictors sampled for tree splits	[1, 9]	
	number of trees in the ensemble	[50, 450]	
	maximum tree depth	[4, 8]	
	proportion of observations sampled for tree splits	0.8**	
	min. loss reduction to allow further tree splits	0**	

*Logistic regression for classification, Gamma GLM with log link for regression
**this hyperparameter is not tuned

Having obtained the best performing models for each training dataset, the corresponding test data was used to calculate the exact Shapley values by fitting models for all possible predictor subsets and calculating the exact Shapley values for predictions on the test set using (5). On the same test data, the Shapley value approximations (M1), (M2), (M5) and the

Table 3: Best models and final hyperparameters for each dataset

Data	Best_Model	Hyperparameter	Value
Rice	GLM	-	-
Seismic bumps	Random Forest	min. number of observations in terminal nodes	13
		number of predictors sampled for tree splits	1
Concrete	XGBoost	learning rate	1.00e-10
		min. number of observations in terminal nodes	4
		number of predictors sampled for tree splits	1
		number of trees in the ensemble	50
		maximum tree depth	4

Table 4: Best models: Performance metrics on test data

Data	Best_Model	Performance_Metric	Value
Rice	GLM	ROC-AuC	0.98
Seismic bumps	Random Forest	ROC-AuC	0.77
Concrete	XGBoost	MAE	2.70

original KernelSHAP version are run and compared to the exact Shapley values to assess the approximation error. The results are given in the next section.

4.3 Comparison of Shapley Value Approximations

For the Rice and Concrete dataset, (M1), (M2), (M5) with default settings were applied along with the KernelSHAP version assuming independence between predictors, from here on called *independence*. In practical settings, one often seeks to reduce the computational burden by sampling a sufficiently large subset of predictors. Thus, all approaches were run with different sample sizes of the predictor combinations to see how sampling a smaller number of unique subsets affects the accuracy of the approximations. When estimating the conditional expectations in (9), M was set to the default 1000 for all approaches. Due to its higher dimensions and the presence of categorical predictors, the only approaches applied on the seismic bumps data were conditional inference trees and the independence assumption, each sampling only 20% of all possible predictor combinations and setting to M to 500 to keep the problem computationally feasible. $v(\emptyset)$ was taken to be $\overline{y}$ in all calculations. All experiments were run once on a conventional laptop (2.7 GHz Dual Core CPU, 8GB RAM) and running times were recorded for all approximation methods.

To evaluate the overall performance of the approximation methods, their mean absolute error with respect to the exact Shapley values was calculated over both observations and predictors. For a more detailed view of the distribution by each predictor, refer to Figures 7-9. The mean absolute errors are compared against the proportion of unique subsets sampled from $\mathcal{X}$ in Figures 10&11, and the recorded running times to assess computational efficiency are found in Figures 12&13. It might be tolerable in some applications to incur a higher approximation error as long as the relative ordering of predictors is preserved. Thus, Fig. 14&15 show the average Kendall's τ between the exact values and their estimators across test observations.

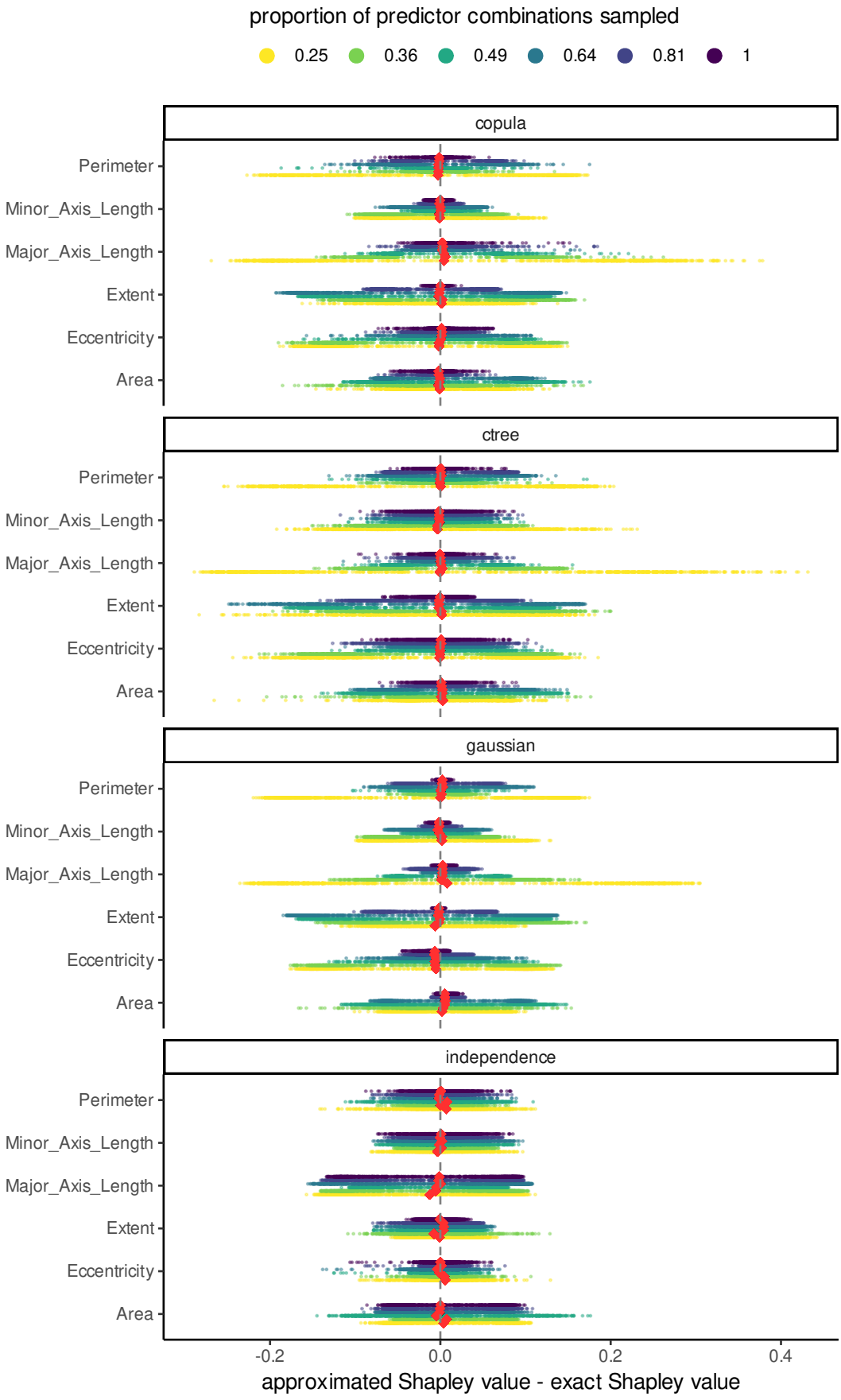

FIGURE 7: Distributions of approximation errors: Rice data (red dots indicate the average)

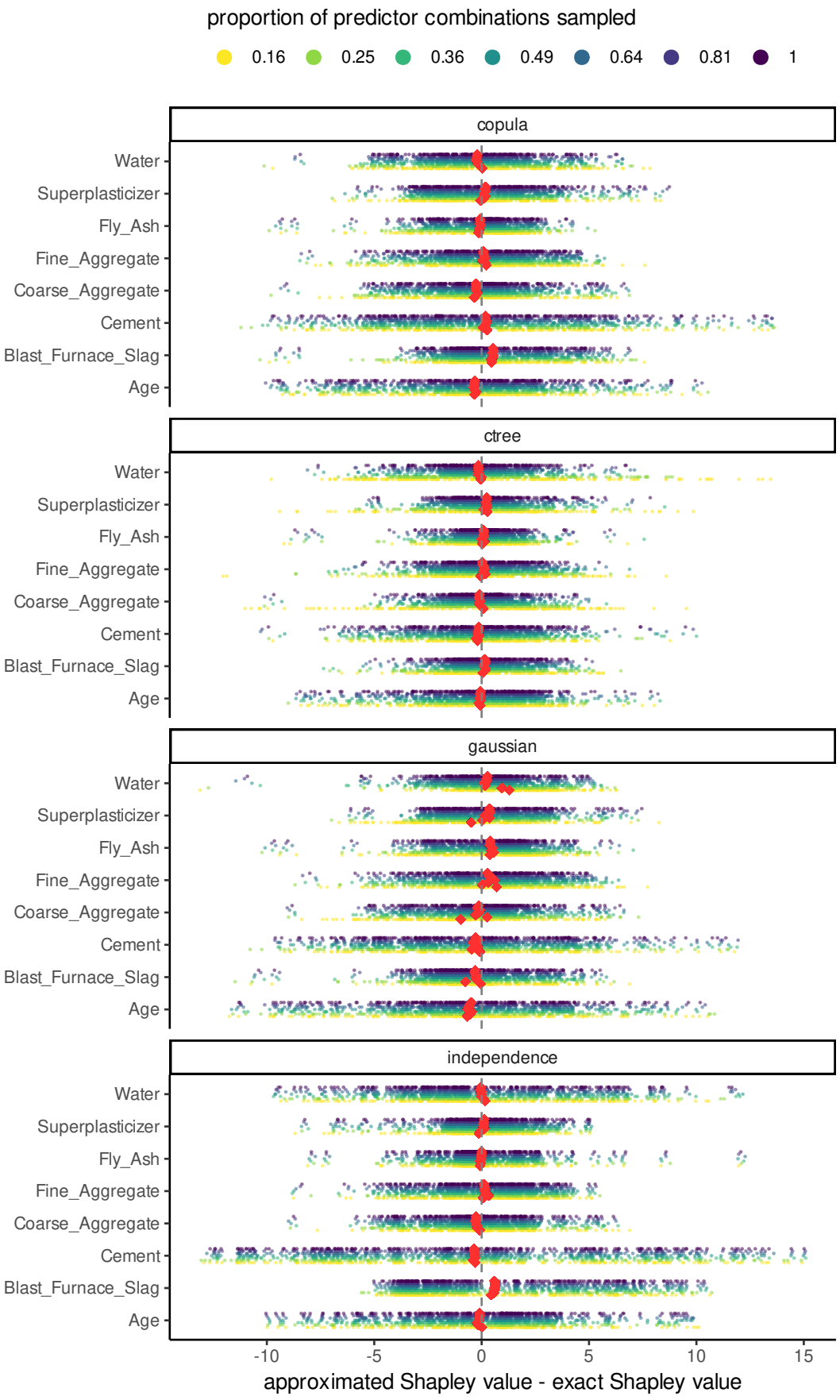

FIGURE 8: Distributions of approximation errors: Concrete data (red dots indicate the average)

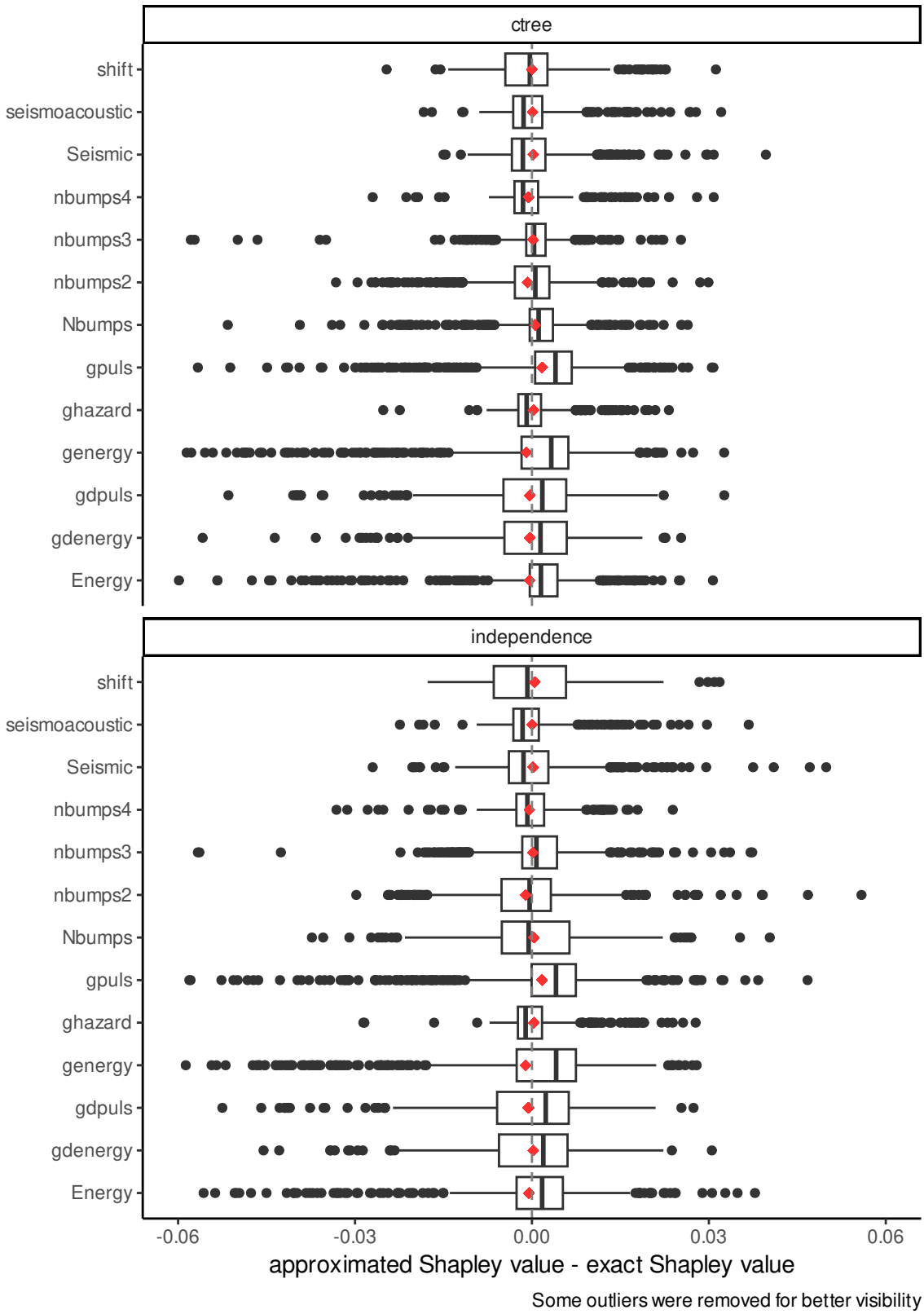

FIGURE 9: Distributions of approximation errors: Seismic data (red dots indicate the average)

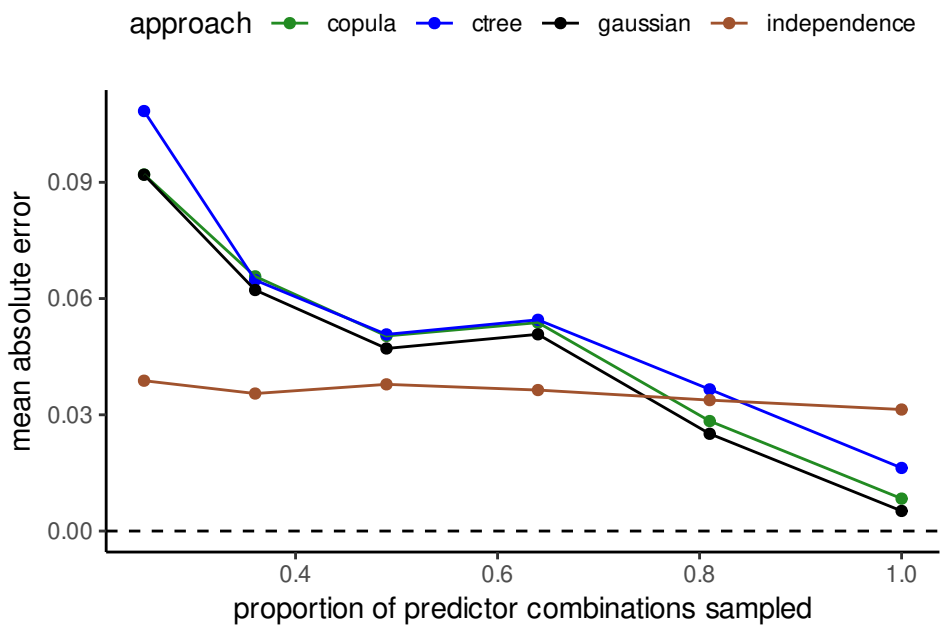

FIGURE 10: Convergence of the approximations: Rice data

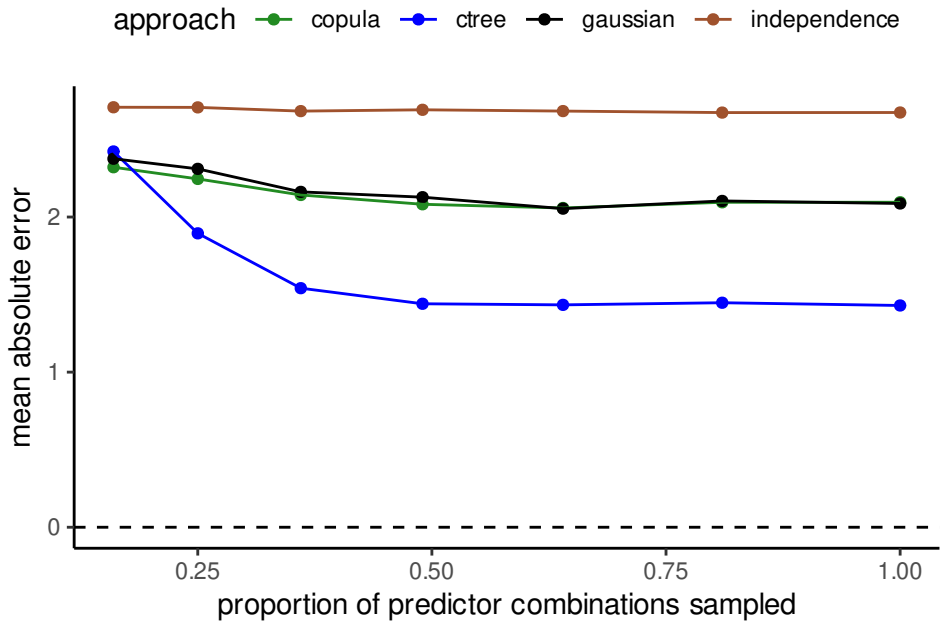

FIGURE 11: Convergence of the approximations: Concrete data

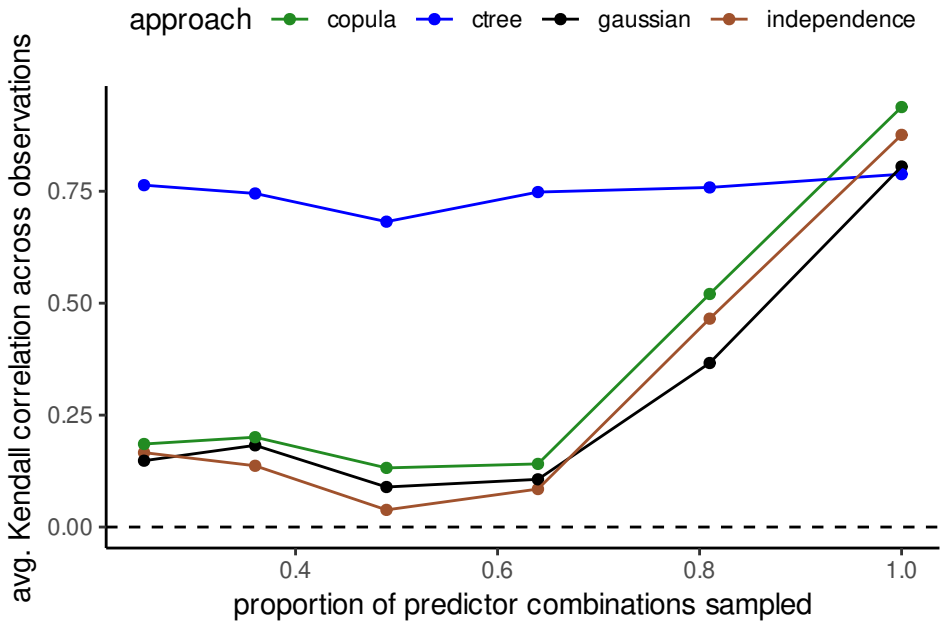

FIGURE 12: Ranking ability of approximations: Rice data

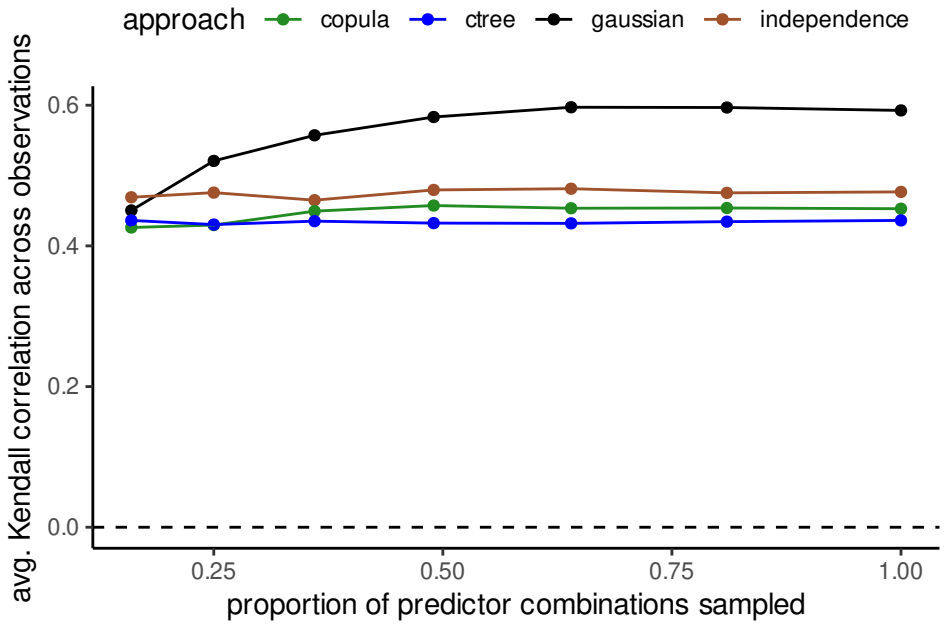

FIGURE 13: Ranking ability of approximations: Concrete data

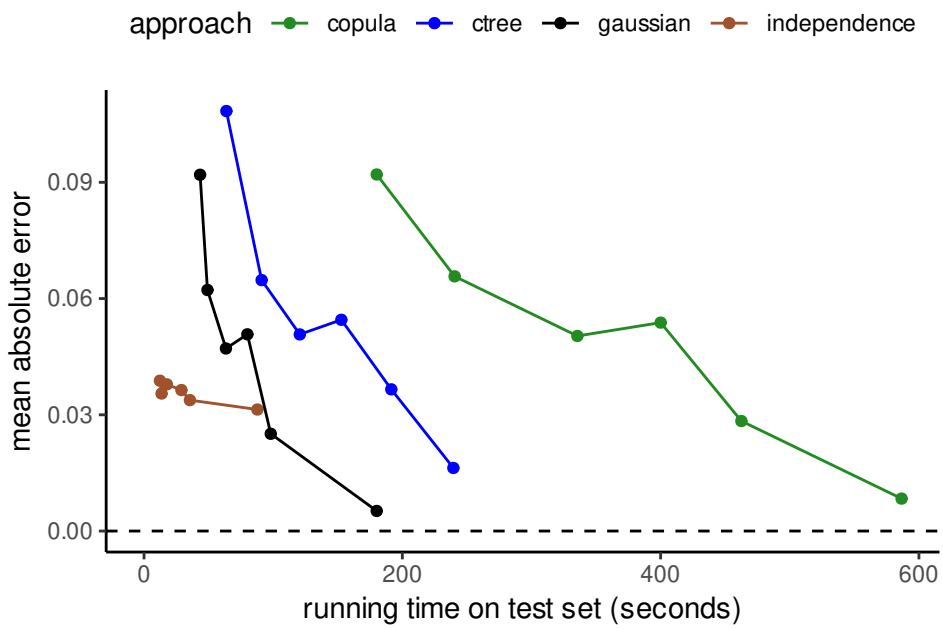

FIGURE 14: Running times of approximations: Rice data

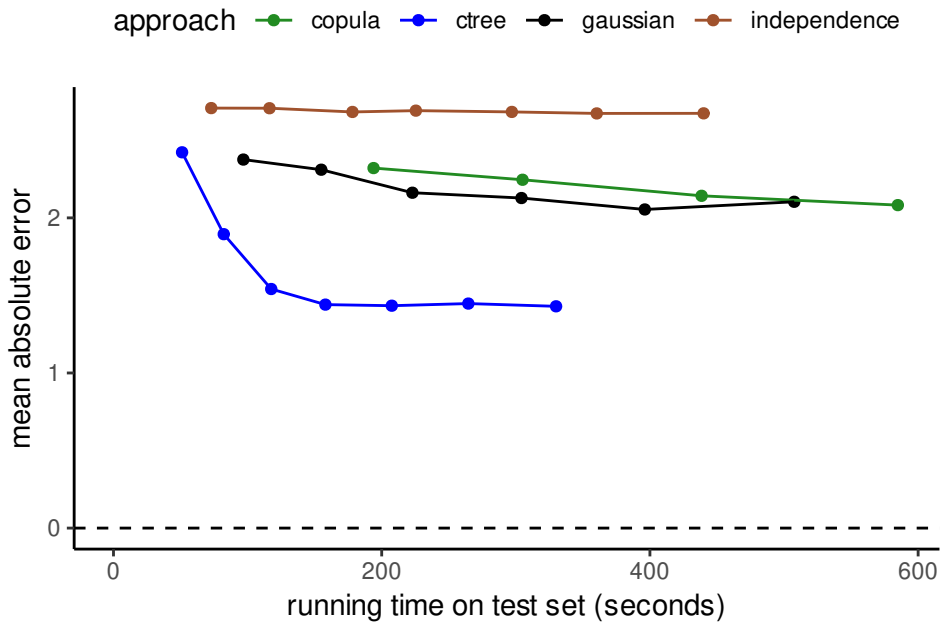

FIGURE 15: Running times of approximations: Concrete data

As expected by looking at the marginal distributions in Fig. 1, the Gaussian and Gaussian copula approaches perform best on the rice data once a large part of the possible predictor combinations $C \subseteq \mathcal{X}$ are sampled. However, when sampling smaller subsets, all approaches except the original KernelSHAP quickly deteriorate in approximation performance. For both the rice and concrete dataset, the independence approach does not exhibit any dependence on

the number of predictor subsets sampled when it comes to MAE. In the case of the concrete data, the conditional inference tree approach stands out by clearly outperforming the other methods in terms of MAE and its tradeoff with running time. It also exhibits a lower MAE in the setting involving the seismic bumps data (0.00555 as compared to 0.00673 for classical KernelSHAP)[6], but needing almost thrice the time to run.

Looking at Figures 7-9, there rarely seem to be systematic approximation errors regarding specific predictors. The approximation error seems to be driven by variance, not any systemic bias. Contrasting Fig. 7 with Fig. 8, we see that the approximations on the concrete dataset do not exhibit the strong sensitivity to the number of predictor subsets sampled. This is likely due to the fact the weights in (7) become more unequal across subsets C when the number of predictors k increases, so for large k, one can sample smaller proportions from $\mathcal{P}(\mathcal{X})$ while covering a larger proportion of the total weights in (7).

The results for ranking ability tend to favor conditional inference trees: In the rice dataset, conditional inference trees clearly outperform the other methods except when no sampling of subsets takes place. For the concrete data, the Gaussian method only slightly outperforms the others without much dependence of sampling proportions by any method. For the seismic bumps data, the ranking ability of *ctree* in the space of observational units seems to be clearly superior (an average Kendall's τ of 0.57 against 0.42 for classical KernelSHAP). As opposed to all other methods, the conditional inference tree approach is never clearly dominated by other methods regarding mean absolute error and ranking ability. Since it can also handle all types of predictors, this method compares most favorably to the others, but is not the most computationally efficient one. For any practical application, increasing this efficiency is crucial in making (M1)-(M5) competitive with their independence-assuming predecessor. Some ideas to do so are discussed in the coming chapter.

[6]These values are so low in an absolute sense due to the imbalance in the target variable, so the characteristic function usually takes on very small values.

5. Discussion of Further Improvements

Since sampling a higher number of predictor combinations is computationally costly but approximation accuracy depends on this number, a possible improvement may lie in the sampling strategy to obtain subsets $S \subset \mathcal{X}$. So far, we have only incorporated the weights in (7) to increase efficiency of KernelSHAP estimators. Other literature has also focused on optimally sampling informative sets of predictor combinations to reduce the variance of the Monte Carlo estimator (9) and thus speeding up its convergence. For instance, Mitchell et al. (2022) consider a variety of techniques to optimally sample permutations of $\mathcal{X}$. Covert & Lee (2021) propose a paired sampling strategy inspired by antithetic sampling: By always selecting a predictor subset C along with its complement $\mathcal{X} \setminus C$, they report large decreases in variance of the classical KernelSHAP estimator. Recently, Goldwasser & Hooker (2023) suggest an approach based on control variates.

In summary, possible improvements to (M1)-(M5) could be made along three dimensions. While the implementation of these approaches is beyond the scope of this thesis, they may point in interesting directions for future research:

(1) Research on optimal sampling strategies is focusing on classical KernelSHAP estimation. Combining the approaches discussed in section 3.1.1 with the recently developed sampling strategies discussed above could accelerate the convergence of (M1)-(M5) and make these approaches more practically feasible, requiring fewer samples to reach a given accuracy level.

(2) Another possibility is to study not the optimal sampling of weighted subsets themselves, but to also investigate properties of the characteristic function. Consider the formula for exact Shapley values from equation (5). Noting that characteristic functions in the explainability setting are usually very different from those considered in game theory (e.g., including many players with negative marginal contributions), one may find a few general properties which were not yet considered in previous work. For example, when predictors carry some information about the target variable and are not independent, marginal contributions to big predictor subsets would tend to be lower than for small subsets, since any information carried by a predictor is more likely to be already captured in a big subset than in a small one. In an importance sampling framework, this would imply an even stronger oversampling of small predictor subsets than already implied by the weights in (5). To incorporate importance weights also based on marginal contributions, one may turn to ideas from information theory. For instance, Peng et al. (2005) propose a procedure for best subset selection based on mutual information. Their idea is to select predictor subsets with both high relevance (i.e. high dependence with the target variable) and low redundancy (i.e. low dependence among predictors). Peng et al. define relevance and redundancy using measures based on mutual information, which has the advantage of not relying on the assumption of linear or monotonic dependencies such as conventional correlation measures. One could apply a similar idea to determining sampling weights for predictor subsets in the Monte Carlo estimation of (5): By giving higher weights to marginal contributions where a predictor X_j to be removed from subset C has low redundancy with C but

high relevance for the target Y, the distribution of marginal contributions could be covered more efficiently. This is because removing a highly relevant predictor from a model with low redundancy is expected to change the model predictions more than in cases with low relevance and high redundancy. Since the estimation of mutual information requires some form of nonparametric density estimation, one could instead use distance correlation (Székely et al., 2007), which is also agnostic to the type of dependencies, but less cumbersome to compute. However, all these ideas would still have to be carried over to the KernelSHAP approximation method, since (7) does not include marginal contributions directly. A starting point for studying properties of the marginal contributions and how to carry over this information to the KernelSHAP framework could be the calculation of exact Shapley values as conducted for chapter 4.

(3) Further work can be done in searching ways how to generate realistic data in order to simulate absence of predictors in order to use the computationally attractive KernelSHAP framework. For instance, Aas, Nagler, et al. (2021) suggest another approach based on vine copulas, which incurs much higher computation times than any of the methods discussed in this work. One may discuss the use of other copulae to incorporate a wider range of dependence structures, but this increased flexibility might come with an increased computational burden. Moreover, the problem of categorical predictors remains. With conditional inference trees, there seems to be a preferable alternative to copula-based methods already.

Moreover, the approximation error of all methods discussed seems to be driven by variability, not bias. To reduce the overall approximation error, it would then be natural to suggest the introduction of a regularization term in (7). When the focus is on explaining individual predictions, a small bias might also be preferred to high variance, especially when the interest lies mainly in a (signed) relative ranking of predictors. A drawback would be the violation of axioms (at least (A1)) on which the use of Shapley values in exlainability was justified originally.

6. Conclusions

In this thesis, recent improvements of Shapley value approximations were discussed and tested on real datasets. Specifically, the approaches suggested by Aas, Jullum, et al. (2021) and Redelmeier et al. (2020) incorporate dependencies between predictors into *KernelSHAP*, the state of the art approximation method which unrealistically assumes mutual independence. As opposed to the aforementioned papers, this work does not rely on simulation studies to assess the methods' performance. Instead, exact calculation of Shapley values on real-world datasets were performed to establish a baseline to compare the approximations to. This allows for testing these approaches in somewhat more realistic settings, which were not the focus of the studies mentioned above. While the approach to testing the methods differs, the empirical results of this work are similar to the aforementioned studies: The improved KernelSHAP estimators of Aas, Jullum, et al. (2021) and Redelmeier et al. (2020) were, with few exceptions, more accurate than the original version of Lundberg & Lee (2017). Due to its generality of handling all predictor types and an overall good performance, the conditional inference tree approach suggested by Redelmeier et al. (2020) seems to compare most favorably in practice.

While the experiments in chapter 4 of this thesis as well as the simulation studies in Aas, Jullum, et al. (2021) and Redelmeier et al. (2020) indicate that their estimators yield improved accuracy, these approaches are also more computationally demanding. The aforementioned papers optimize the procedure of how to accurately simulate absence of predictors in a model. To decrease running times under computational constraints or to further improve accuracy, one can also focus on the procedure of sampling optimal predictor combinations. To this end, recent work has focused on sampling strategies to reduce the variance of the classical KernelSHAP estimator. Combining the modeling of predictor dependencies with such refined sampling approaches might improve the Monte Carlo estimates further. Some suggestions for further research in these directions were thus given in the previous chapter.

However, with Shapley value approximations becoming a complex modeling effort by themselves, one is not escaping the tradeoff between model accuracy and interpretability. With increasingly sophisticated Shapley value approximations, explainability itself becomes opaque and thus defeats its purpose. In small dimensions, it might be more transparent to calculate Shapley values by brute force, as was done in this work.

References

Aas, K., Jullum, M., & Løland, A. (2021) Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artificial Intelligence*, 298.

Aas, K., Nagler, T., Jullum, M., & Løland, A. (2021) Explaining predictive models using shapley values and non-parametric vine copulas. *Dependence Modeling*, 9, 62–81.

Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., & Müller, K.R. (2010) How to explain individual classification decisions. *Journal of Machine Learning Research*, 11, 1803–1831.

Bradley, A.P. (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30, 1145–1159.

Breiman, L. (1996) Stacked regressions. *Machine learning*, 24, 49–64.

Breiman, L. (2001) Random forests. *Machine learning*, 45, 5–32.

Breiman, L., Friedman, J., Stone, C.J., & Olshen, R.A. (1984) *Classification and regression trees*, Taylor & Francis.

Burkart, N. & Huber, M.F. (2021) A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70, 245–317.

Charnes, A., Golany, B., Keane, M., & Rousseau, J. (1988) Extremal principle solutions of games in characteristic function form: Core, chebychev and shapley value generalizations. *Econometrics of planning and efficiency*, Springer Netherlands, pp. 123–133.

Chen, T. & Guestrin, C. (2016) XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*.

Cinar, I. & Koklu, M. (2019) Classification of rice varieties using artificial intelligence methods. *International Journal of Intelligent Systems and Applications in Engineering*, 7, 188–194.

Covert, I. & Lee, S.-I. (2021) Improving KernelSHAP: Practical shapley value estimation via linear regression. *Proceedings of the 24th international conference on artificial intelligence and statistics*, pp. 3457–3465.

Friedman, J.H. (2001) Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29, 1189–1232.

Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2015) Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24, 44–65.

Goldwasser, J. & Hooker, G. (2023) Stabilizing estimates of shapley values with control variates.

Grömping, U. (2015) Variable importance in regression models. *Wiley interdisciplinary reviews: Computational statistics*, 7, 137–152.

Hothorn, T., Hornik, K., & Zeileis, A. (2006) Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15, 651–674.

Jethani, N., Sudarshan, M., Covert, I.C., Lee, S.-I., & Ranganath, R. (2022) FastSHAP: Real-time shapley value estimation. *10th international conference on learning representations*.

Jullum, M., Redelmeier, A., & Aas, K. (2021) groupShapley: Efficient prediction explanation

with shapley values for feature groups.

Kelly, M., Longjohn, R., & Nottingham, K. (2023) *The UCI machine learning repository*.

Kruskal, W. (1987) Relative importance by averaging over orderings. *The American Statistician*, 41, 6–10.

Kuhn, M. & Wickham, H. (2020) *Tidymodels: A collection of packages for modeling and machine learning using tidyverse principles.*

Leeper, T.J. (2017) Interpreting regression results using average marginal effects with r ' s margins.

Lipovetsky, S. & Conklin, M. (2001) Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17, 319–330.

Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020) From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2, 56–67.

Lundberg, S.M. & Lee, S.-I. (2017) A unified approach to interpreting model predictions. *Advances in neural information processing systems*, vol. 30, pp. 4765–4774.

Mann, I. & Shapley, L.S. (1960) *Values of large games, IV: Evaluating the electoral college by montecarlo techniques*, RAND Corporation.

Mitchell, R., Cooper, J., Frank, E., & Holmes, G. (2022) Sampling permutations for shapley value estimation. *Journal of Machine Learning Research*, 23, 2082–2127.

Montavon, G., Kauffmann, J., Samek, W., & Müller, K.-R. (2022) Explaining the predictions of unsupervised learning models. *xxAI - beyond explainable AI: International workshop, held in conjunction with ICML 2020, july 18, 2020, vienna, austria, revised and extended papers*, pp. 117–138.

Nelder, J.A. & McCullagh, P. (1983) *Generalized linear models*, Monographs on statistics and applied probability, Kluwer Academic Publishers.

Niculescu-Mizil, A. & Caruana, R. (2005) Predicting good probabilities with supervised learning. *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, pp. 625–632.

Nisan, N. & Ronen, A. (2001) Algorithmic mechanism design. *Games and Economic Behavior*, 35, 166–196.

Peng, H., Long, F., & Ding, C. (2005) Feature selection based on mutual information: Criteria of max-dependency,max-relevance, and min-redundancy. *IEEE transactions on pattern analysis and machine intelligence*, 27, 1226–1238.

R Core Team (2023) *R: A language and environment for statistical computing*, Vienna, Austria: R Foundation for Statistical Computing.

Redelmeier, A., Jullum, M., & Aas, K. (2020) Explaining predictive models with mixed features using shapley values and conditional inference trees. *Lecture Notes in Computer Science*, 117–137.

Ribeiro, M.T., Singh, S., & Guestrin, C. (2016) "Why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144.

Robnik-Sikonja, M. & Kononenko, I. (2008) Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20, 589–600.

Sellereite, N., Jullum, M., Redelmeier, A., & Lachmann, J. (2024) *Shapr: Prediction explanation with dependence-aware shapley values*.

Shapley, L.S. (1953) A value for n-person games. *Contributions to the theory of games II*, pp. 307–317.

Sikora, M. & Wróbel, Ł. (2010) Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, 55, 91–114.

Sklar, M. (1959) Fonctions de répartition à N dimensions et leurs marges. *Annales de l'ISUP*, 8, 229–231.

Smith, J.M. & Price, G.R. (1973) The logic of animal conflict. *Nature*, 246, 15–18.

Štrumbelj, E. & Kononenko, I. (2010) An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, 11, 1–18.

Sundararajan, M., Taly, A., & Yan, Q. (2017) Axiomatic attribution for deep networks. *Proceedings of the 34th international conference on machine learning*, pp. 3319–3328.

Székely, G.J., Rizzo, M.L., & Bakirov, N.K. (2007) Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35, 2769–2794.

Wooldridge, J.M. (2010) Conditional expectations and related concepts in econometrics. *Econometric analysis of cross section and panel data*, The MIT Press, pp. 13–36.

Yeh, I.-C. (1998) Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28, 1797–1808.

Appendix: R Code

Note: Filepaths were saved as characters beforehand, for example: `path_to_data <- "C:/my/directory"`.

```
# setup
library(tidyverse)
library(tidymodels)
library(readxl)
library(ranger)
library(xgboost)
library(reshape2)
library(kableExtra)
library(shapr)
library(progressr)

seed <- 202401
set.seed(seed)


## DATA PREPROCESSING##

# load raw datasets
seismic_raw <- read.table(path_to_SeismicData, sep=",", header = T)
concrete_raw <- readxl::read_excel(path_to_ConcreteData)
rice_raw <- read.table(path_to_RiceData, sep=",", header = T)


# datasets ready for test/train split----
seismic <- seismic_raw %>% mutate(across(where(is.character), ~as.factor(.))) %>%
  rename(target=class, Seismic=seismic, Nbumps=nbumps, Energy=energy) %>%
  # upper cases avoid problems with string matching later
  mutate(target=as.factor(target))

colnames(concrete_raw) <- gsub("\\(.*", "", colnames(concrete_raw))
colnames(concrete_raw) <- trimws(colnames(concrete_raw))
colnames(concrete_raw) <- sub(" ", "_", colnames(concrete_raw))
colnames(concrete_raw) <- sub(" ", "_", colnames(concrete_raw))

concrete <- concrete_raw %>% rename(target=ncol(concrete_raw))
# last column is the target variable

rice <- rice_raw %>% rename(target=Class) %>% mutate(target=as.factor(target))

# convert factors in numeric for graphics
seismic_num <- seismic %>% mutate(across(where(is.factor), ~as.numeric(.)))
concrete_num <- concrete %>% mutate(across(where(is.factor), ~as.numeric(.)))
rice_num <- rice %>% mutate(across(where(is.factor), ~as.numeric(.)))

# data preparation for modeling----
```

```r
# remove predictors with extremely high correlation
# and those who (almost) only take one value
seismic <- seismic %>% select(-c(nbumps5, nbumps6, nbumps7, nbumps89, maxenergy))
rice <- rice %>% select(-Convex_Area)


# set up train / test splits & cross-validation----

# proportion of data for training
train_prop <- 0.8

# number of folds for cross-validation
k_folds <- 5

# test/train split & CV
set.seed(seed)

seismic_split <- initial_split(seismic, prop = train_prop)
seismic_train <- training(seismic_split)
seismic_test <- testing(seismic_split)
seismic_trainCV <- vfold_cv(seismic_train, v=k_folds)

concrete_split <- initial_split(concrete, prop = train_prop)
concrete_train <- training(concrete_split)
concrete_test <- testing(concrete_split)
concrete_trainCV <- vfold_cv(concrete_train, v=k_folds)

rice_split <- initial_split(rice, prop = train_prop)
rice_train <- training(rice_split)
rice_test <- testing(rice_split)
rice_trainCV <- vfold_cv(rice_train, v=k_folds)


# model training----

# define the classification models
logit_mod <- logistic_reg(
  engine = "glm"
)

rf_mod <- rand_forest(
  mode = "classification",
  engine = "ranger",
  trees = 500,
  mtry = tune(),
  min_n = tune()
)

xgb_mod <- boost_tree(
  mode = "classification",
```

```
  engine = "xgboost",
  sample_size = 0.8,
  learn_rate=tune(),
  min_n=tune(),
  mtry=tune(),
  trees=tune(),
  tree_depth=tune()
)


# define the regression models
gamma_mod <- linear_reg() %>%
  set_engine("glm", family = Gamma(link = "log"))

rf_mod_reg <- rf_mod %>% set_mode("regression")

xgb_mod_reg <- xgb_mod %>% set_mode("regression")


# set up hyperparameter search grids----

grid_rf <- grid_regular(
  mtry(range = c(1, 10)),
  min_n(range = c(5, 14)),
  levels = 10
)

grid_xgb <- grid_regular(
  learn_rate(),
  min_n(range = c(4, 20)),
  mtry(range = c(1, 9)),
  trees(range = c(50, 450)),
  tree_depth(range = c(4, 8)),
  levels = 5
)

# options
ctrl <- control_resamples(verbose = T, save_pred = F,
                          event_level = "second", parallel_over = "everything")


# create workflow objects----

# define model formulae & datasets
seismic_rec <- recipe(target ~., data=seismic_train)
concrete_rec <- recipe(target ~., data=concrete_train)
rice_rec <- recipe(target ~., data=rice_train)

# prepare hyperparameter tuning
all_workflows_seismic <- workflow_set(
```

```
    preproc = list("seismic" = seismic_rec),
    models = list(logit = logit_mod, rf = rf_mod)) %>%
  option_add(id = "seismic_logit", control = ctrl) %>%
  option_add(id = "seismic_rf", control = ctrl, grid = grid_rf)

all_workflows_rice <- workflow_set(
    preproc = list("rice" = rice_rec),
    models = list(logit = logit_mod, rf = rf_mod, xgb=xgb_mod)) %>%
  option_add(id = "rice_logit", control = ctrl) %>%
  option_add(id = "rice_rf", control = ctrl, grid = grid_rf) %>%
  option_add(id = "rice_xgb", control=ctrl, grid=grid_xgb)

all_workflows_concrete <- workflow_set(
  preproc = list("concrete" = concrete_rec),
  models = list(gamma = gamma_mod, rf = rf_mod_reg, xgb=xgb_mod_reg)) %>%
  option_add(id = "concrete_gamma", control = ctrl) %>%
  option_add(id = "concrete_rf", control = ctrl, grid = grid_rf) %>%
  option_add(id = "concrete_xgb", control = ctrl, grid=grid_xgb)


# model tuning----

#seismic data
tuning_results_seismic <- all_workflows_seismic %>%
  workflow_map(resamples = seismic_trainCV, verbose = T, seed = seed,
               metrics = metric_set(mn_log_loss))
# save results to work with later
save(tuning_results_seismic, file = path_wf_seismic)

# rice data
tuning_results_rice <- all_workflows_rice %>%
  workflow_map(resamples = rice_trainCV, verbose = T, seed = seed,
               metrics = metric_set(mn_log_loss))
# save results to work with later
save(tuning_results_rice, file = path_wf_rice)

# concrete data
tuning_results_concrete <- all_workflows_concrete %>%
  workflow_map(resamples = concrete_trainCV, verbose = T, seed = seed,
               metrics = metric_set(rmse))
# save results to work with later
save(tuning_results_concrete, file = path_wf_concrete)


#load results
load(path_wf_seismic)
load(path_wf_rice)
load(path_wf_concrete)
```

```r
# see what models performed best

tuning_params <- c("mtry", "min_n", "learn_rate", "min_n", "mtry",
                   "trees", "tree_depth")
tuning_params_seismic <- c("mtry", "min_n")

best_model_seismic <- tuning_results_seismic %>%
  select(wflow_id, result) %>% unnest(result) %>%
  select(-c(splits, .notes)) %>% unnest(.metrics) %>%
  select(-c(id, .config, .estimator)) %>%
  group_by(wflow_id, .metric, across(all_of(tuning_params_seismic))) %>%
  summarise(mean_logloss=mean(.estimate)) %>% ungroup() %>%
  arrange(mean_logloss) %>%
  slice_head()

best_model_rice <- tuning_results_rice %>%
  select(wflow_id, result) %>% unnest(result) %>%
  select(-c(splits, .notes)) %>% unnest(.metrics) %>%
  select(-c(id, .config, .estimator)) %>%
  group_by(wflow_id, .metric, across(all_of(tuning_params))) %>%
  summarise(mean_logloss=mean(.estimate)) %>% ungroup() %>%
  arrange(mean_logloss) %>%
  slice_head()

best_model_concrete <- tuning_results_concrete %>%
  select(wflow_id, result) %>% unnest(result) %>%
  select(-c(splits, .notes)) %>% unnest(.metrics) %>%
  select(-c(id, .config, .estimator)) %>%
  group_by(wflow_id, .metric, across(all_of(tuning_params))) %>%
  summarise(mean_rmse=mean(.estimate)) %>% ungroup() %>%
  arrange(-mean_rmse) %>%
  slice_head()


# fit best models on the whole training set, get performance metrics on test set----
best_seismic <-
  tuning_results_seismic %>%
  extract_workflow_set_result("seismic_rf") %>%
  select_best(metric = "mn_log_loss")

seismic_last_fit <-
  tuning_results_seismic %>%
  extract_workflow("seismic_rf") %>%
  finalize_workflow(best_seismic) %>%
  last_fit(split = seismic_split)

seismic_test_preds <- seismic_last_fit %>% collect_predictions()


best_rice <-
```

```
  tuning_results_rice %>%
  extract_workflow_set_result("rice_logit") %>%
  select_best(metric = "mn_log_loss")

rice_last_fit <-
  tuning_results_rice %>%
  extract_workflow("rice_logit") %>%
  finalize_workflow(best_rice) %>%
  last_fit(split = rice_split)

rice_test_preds <- rice_last_fit %>% collect_predictions()


best_concrete <-
  tuning_results_concrete %>%
  extract_workflow_set_result("concrete_xgb") %>%
  select_best(metric = "rmse")

concrete_last_fit <-
  tuning_results_concrete %>%
  extract_workflow("concrete_xgb") %>%
  finalize_workflow(best_concrete) %>%
  last_fit(split = concrete_split)

concrete_test_preds <- concrete_last_fit %>% collect_predictions()


# true shapley values by brute force computation----

# set expected predictions of null model (mean of target on train data)
nullmod_pred_seismic <- mean(as.numeric(seismic_train$target)-1)
nullmod_pred_rice <- mean(as.numeric(rice_train$target)-1)
nullmod_pred_concrete <- mean(concrete_train$target)


# extract fits from model engines
engine_seismic <- seismic_last_fit %>% extract_fit_engine()
engine_rice <- rice_last_fit %>% extract_fit_engine()
engine_concrete <- concrete_last_fit %>% extract_fit_engine()


# predictor names
pred_names_seismic <- names(seismic_test%>% select(-target))
pred_names_rice <- names(rice_test %>% select(-target))
pred_names_concrete <- names(concrete_test %>% select(-target))

#number of predictors
k_rice <- length(pred_names_rice)
k_concrete <- length(pred_names_concrete)
k_seismic <- length(pred_names_seismic)
```

```r
# all possible predictor combinations
pred_combs_seismic <- unlist(lapply(1:length(pred_names_seismic),
                function(b) combn(1:k_seismic,b,simplify = F)), recursive = F)
pred_combs_rice <- unlist(lapply(1:length(pred_names_rice),
                function(b) combn(1:k_rice,b,simplify = F)), recursive = F)
pred_combs_concrete <- unlist(lapply(1:length(pred_names_concrete),
                function(b) combn(1:k_concrete,b,simplify = F)), recursive = F)

# predictor combinations as character vector
pred_char_seismic <- sapply(pred_combs_seismic, function(b)
  paste(pred_names_seismic[b], collapse = ""))
pred_char_rice <- sapply(pred_combs_rice, function(b)
  paste(pred_names_rice[b], collapse = ""))
pred_char_concrete <- sapply(pred_combs_concrete, function(b)
  paste(pred_names_concrete[b], collapse = ""))

# all possible model formulas
formulas_seismic <- sapply(pred_combs_seismic, function(b)
  paste("target ~", paste(pred_names_seismic[b], collapse = "+")))
formulas_rice <- sapply(pred_combs_rice, function(b)
  paste("target ~", paste(pred_names_rice[b], collapse = "+")))
formulas_concrete <- sapply(pred_combs_concrete, function(b)
  paste("target ~", paste(pred_names_concrete[b], collapse = "+")))


# fit all possible models, get predictions on train&test data----

# fits all 2^k-1 possible models for each dataset

# final models that include tuned hyperparameters
seismic_mod_final <- tuning_results_seismic %>%
  extract_workflow("seismic_rf") %>%
  finalize_workflow(best_seismic) %>%
  extract_spec_parsnip()

concrete_mod_final <- tuning_results_concrete %>%
  extract_workflow("concrete_xgb") %>%
  finalize_workflow(best_concrete) %>%
  extract_spec_parsnip()


# rice data
rice_pset_preds <- matrix(nrow = nrow(rice_test), ncol = length(formulas_rice))

for(m in 1:length(formulas_rice)){
  mod <- glm(formula = formulas_rice[m], family = binomial, data = rice_train)
  rice_pset_preds[, m] <- predict.glm(mod, newdata = rice_test, type = "response")
}

#append null model predictions
```

```r
rice_pset_preds_final <- cbind(rep(nullmod_pred_rice, nrow(rice_test)),
                               rice_pset_preds)%>% as.data.frame()

colnames(rice_pset_preds_final) <- c("nullmodel_preds", pred_char_rice)


# concrete data
concrete_pset_preds <- matrix(nrow = nrow(concrete_test),
                              ncol = length(formulas_concrete))

for(m in 1:length(formulas_concrete)){
  mod <- parsnip::fit(concrete_mod_final,
                      as.formula(formulas_concrete[m]), data=concrete_train)
  concrete_pset_preds[, m] <- as.matrix(predict(mod, new_data = concrete_test))
}

#append null model predictions
concrete_pset_preds_final <- cbind(rep(nullmod_pred_concrete, nrow(concrete_test)),
                                   concrete_pset_preds)%>% as.data.frame()

colnames(concrete_pset_preds_final) <- c("nullmodel_preds", pred_char_concrete)


# seismic data
seismic_pset_preds <- matrix(nrow = nrow(seismic_test), ncol = length(formulas_seismic))

for(m in 1:length(formulas_seismic)){
  mod <- parsnip::fit(seismic_mod_final, as.formula(formulas_seismic[m]),
                      data=seismic_train)
  seismic_pset_preds[, m] <- as.matrix(predict(mod, new_data = seismic_test,
                                               type = "prob")[,2])
}

#append null model predictions
seismic_pset_preds_final <- cbind(rep(nullmod_pred_seismic, nrow(seismic_test)),
                                  seismic_pset_preds)%>% as.data.frame()

colnames(seismic_pset_preds_final) <- c("nullmodel_preds", pred_char_seismic)


# Shapley weights----

# using definition where player j is removed
shapley_weights <- function(test_dat){

  wts <- c()
  subset_size <- c()
  k <- ncol(test_dat %>% select(-target))

  for(b in 1:k){
```

```
    wt<-rep(factorial(b-1)*factorial(k-b), choose(k-1, b-1))
    wts<- c(wts, wt)
    subset_size <- c(subset_size, rep(b, choose(k-1, b-1)))
  }
  shapley_wts <- cbind(wts, subset_size)
}

shapley_wts_rice <- shapley_weights(rice_test)
shapley_wts_seismic <- shapley_weights(seismic_test)
shapley_wts_concrete <- shapley_weights(concrete_test)


# calculate the exact Shapley values----

exact_shapleys <- function(test_dat, powerset_preds, shapley_wts){

  k <- ncol(test_dat %>% select(-target))
  pred_names <- colnames(test_dat %>% select(-target))

  shapleys_exact <- matrix(nrow = nrow(test_dat), ncol = k)

  for(j in 1:k){

    # select model predictions where formula contains predictor j
    pred_j <- pred_names[j]
    predicted_j <- as.data.frame(powerset_preds) %>%
      select((contains(pred_j, ignore.case = F)))
    predsets_j <- colnames(predicted_j)

    # remove predictor j
    predsets_j_removed <- gsub(pred_j, "", predsets_j)
    # marginal contributions to empty sets
    predsets_j_removed <- ifelse(predsets_j_removed!="",
                                 predsets_j_removed, "nullmodel_preds")

    predicted_j_removed <- powerset_preds %>% as.data.frame() %>%
      select(all_of(predsets_j_removed))

    # matrix of marginal contributions (columns are already in the right order)
    marg_contribs <- as.matrix(predicted_j-predicted_j_removed)

    # vectors of exact shapley values
    shapleys_exact[, j] <- (marg_contribs%*%as.matrix(shapley_wts)[,1])/factorial(k)
  }

  colnames(shapleys_exact) <- pred_names
  return(shapleys_exact)
}
```

```r
shapleys_exact_rice <- exact_shapleys(rice_test, rice_pset_preds_final,
                                      shapley_wts_rice)
shapleys_exact_seismic <- exact_shapleys(seismic_test, seismic_pset_preds_final,
                                         shapley_wts_seismic)
shapleys_exact_concrete <- exact_shapleys(concrete_test, concrete_pset_preds_final,
                                          shapley_wts_concrete)


# save the exact shapley values
save(shapleys_exact_rice, file=path_shapley_exact_rice)
save(shapleys_exact_seismic, file=path_shapley_exact_seismic)
save(shapleys_exact_concrete, file=path_shapley_exact_concrete)



# Shapley Value approximations----

# using the package of Jullum et al
# approach="independence" corresponds to original KernelSHAP

approx_shapr <- function(model_eng, test_dat, train_dat,
                         approach, pred_zero, prop_combs){

  shap_shapr <- shapr::explain(model_eng,
                               x_explain=(test_dat %>% select(-target)),
                               x_train=(train_dat %>% select(-target)),
                               approach=approach, prediction_zero=pred_zero,
                               n_combinations = floor(prop_combs*2^(ncol(test_dat)-1)),
                               n_samples = 1000,
                               seed = seed, timing = T
  )

  runtime <- rep(shap_shapr[[5]]$total_time_secs, nrow(test_dat))
  combs_prop <- rep(prop_combs, nrow(test_dat))

  shap_shapr <- cbind(shap_shapr$shapley_values[, -1], combs_prop, runtime)

  return(shap_shapr)
}

approx_shapr_samplecombs <- function(model_eng, test_dat, train_dat,
                                     approach, pred_zero, prop_combs){

  shapr_approx <- matrix(nrow = nrow(test_dat)*length(prop_combs),
                         ncol = ncol(test_dat)+1)

  for(i in 1:length(prop_combs)){
    shapr_approx[((i-1)*nrow(test_dat)+1):(i*nrow(test_dat)), ] <- as.matrix(
                                approx_shapr(
                                   model_eng=model_eng, test_dat=test_dat,
```

```
                                    train_dat=train_dat, approach=approach,
                                    pred_zero = pred_zero, prop_combs = prop_combs[i]))

  }
  shapr_approx <- as.data.frame(shapr_approx) %>% mutate(Approach=approach)
  colnames(shapr_approx) <- c(colnames(test_dat%>%select(-target)),
                              "combs_prop", "runtime", "approach")
  return(shapr_approx)
}



# rice data
pset_prop_rice <- seq(0.5, 1, by=0.1)^2
gc()
progressr::handlers(global = TRUE)

approx_shapr_rice_indep <- approx_shapr_samplecombs(engine_rice, rice_test, rice_train,
                         "independence", nullmod_pred_rice, prop_combs = pset_prop_rice)
save(approx_shapr_rice_indep, file = paste0(tuning_results_path, "/shapr_rice_indep.RData"))
gc()
approx_shapr_rice_gaussian <- approx_shapr_samplecombs(engine_rice, rice_test, rice_train,
                           "gaussian", nullmod_pred_rice, prop_combs = pset_prop_rice)
save(approx_shapr_rice_gaussian, file = paste0(tuning_results_path,
                                               "/shapr_rice_gaussian.RData"))
gc()
approx_shapr_rice_copula <- approx_shapr_samplecombs(engine_rice, rice_test, rice_train,
                                                     "copula", nullmod_pred_rice,
                                                     prop_combs = pset_prop_rice)
save(approx_shapr_rice_copula, file = paste0(tuning_results_path,
                                             "/shapr_rice_copula.RData"))
gc()
approx_shapr_rice_ctree <- approx_shapr_samplecombs(engine_rice, rice_test, rice_train,
                                                    "ctree", nullmod_pred_rice,
                                                  prop_combs = pset_prop_rice)
save(approx_shapr_rice_ctree, file = paste0(tuning_results_path,
                                            "/shapr_rice_ctree.RData"))
gc()



#concrete data
pset_prop_concrete <- seq(0.4, 1, by=0.1)^2
gc()
progressr::handlers(global = TRUE)

approx_shapr_concrete_indep <- approx_shapr_samplecombs(
  engine_concrete, concrete_test, concrete_train, "independence",
  nullmod_pred_concrete, prop_combs=pset_prop_concrete)
save(approx_shapr_concrete_indep,
```

```
     file = paste0(tuning_results_path, "/shapr_concrete_indep.RData"))
gc()
approx_shapr_concrete_gaussian <- approx_shapr_samplecombs(
  engine_concrete, concrete_test, concrete_train,
    "gaussian", nullmod_pred_concrete, prop_combs = pset_prop_concrete)
save(approx_shapr_concrete_gaussian,
     file = paste0(tuning_results_path, "/shapr_concrete_gaussian.RData"))
gc()
approx_shapr_concrete_copula <- approx_shapr_samplecombs(
  engine_concrete, concrete_test, concrete_train,
    "copula", nullmod_pred_concrete, prop_combs = pset_prop_concrete)
save(approx_shapr_concrete_copula,
     file = paste0(tuning_results_path, "/shapr_concrete_copula.RData"))
gc()
approx_shapr_concrete_ctree <- approx_shapr_samplecombs(
  engine_concrete, concrete_test, concrete_train,
    "ctree", nullmod_pred_concrete, prop_combs = pset_prop_concrete)
save(approx_shapr_concrete_ctree, file = paste0(tuning_results_path, "/shapr_concrete_ctree.RData"
gc()



#seismic data
gc()
progressr::handlers(global = TRUE)

# loop over observations manually to avoid memory clogging
# can alternatively use the function 'approx_shapr_samplecombs'

# original KernelSHAP
approx_shapr_seismic_indep <- matrix(nrow = nrow(seismic_test),
                                     ncol = (ncol(seismic_test)+1))

for(i in 1:nrow(seismic_test)){
  approx_shapr_seismic_indep[i, 1:(ncol(seismic_test)+1)] <- as.matrix(
    approx_shapr(engine_seismic, seismic_test[i, ], seismic_train,
                 "independence", nullmod_pred_seismic, prop_combs = 0.2))
  gc()
}

approx_shapr_seismic_indep <- bind_cols(as.data.frame(
    approx_shapr_seismic_indep), rep("independence", nrow(seismic_test)))
colnames(approx_shapr_seismic_indep) <- c(colnames(
    seismic_test%>%select(-target)),
      "combs_prop", "runtime", "approach")
save(approx_shapr_seismic_indep, file = paste0(tuning_results_path,
                                               "/shapr_seismic_indep.RData"))
gc()
```

```r
# ctrees
approx_shapr_seismic_ctree <- matrix(nrow = nrow(seismic_test),
                                     ncol = (ncol(seismic_test)+1))

for(i in 1:nrow(seismic_test)){
  approx_shapr_seismic_ctree[i, 1:(ncol(seismic_test)+1)] <- as.matrix(
    approx_shapr(engine_seismic, seismic_test[i, ], seismic_train,
           "ctree", nullmod_pred_seismic, prop_combs = 0.2))
  gc()
}

approx_shapr_seismic_ctree <- bind_cols(as.data.frame(
  approx_shapr_seismic_ctree), rep("ctree", nrow(seismic_test)))
colnames(approx_shapr_seismic_ctree) <- c(colnames(seismic_test%>%select(-target)),
                                  "combs_prop", "runtime", "approach")
save(approx_shapr_seismic_ctree, file = paste0(tuning_results_path,
                                               "/shapr_seismic_ctree.RData"))
gc()

# load exact shapley values
load(path_shapley_exact_rice)
load(path_shapley_exact_seismic)
load(path_shapley_exact_concrete)


# load results from the approximations
load(paste0(tuning_results_path, "/shapr_rice_indep.RData"))
load(paste0(tuning_results_path, "/shapr_rice_gaussian.RData"))
load(paste0(tuning_results_path, "/shapr_rice_copula.RData"))
load(paste0(tuning_results_path, "/shapr_rice_ctree.RData"))
load(paste0(tuning_results_path, "/shapr_concrete_indep.RData"))
load(paste0(tuning_results_path, "/shapr_concrete_gaussian.RData"))
load(paste0(tuning_results_path, "/shapr_concrete_copula.RData"))
load(paste0(tuning_results_path, "/shapr_concrete_ctree.RData"))
load(paste0(tuning_results_path, "/shapr_seismic_indep.RData"))
load(paste0(tuning_results_path, "/shapr_seismic_ctree.RData"))

# for each dataset, put results of different methods together for plotting
approx_shapr_rice <- bind_rows(approx_shapr_rice_indep,
                               approx_shapr_rice_gaussian,
                               approx_shapr_rice_copula,
                               approx_shapr_rice_ctree)
approx_shapr_concrete <- bind_rows(approx_shapr_concrete_indep,
                                   approx_shapr_concrete_gaussian,
                                   approx_shapr_concrete_copula,
                                   approx_shapr_concrete_ctree)
approx_shapr_seismic <- bind_rows(approx_shapr_seismic_indep,
                                  approx_shapr_seismic_ctree)
```

```r
# proportions of unique predictor combinations
# sampled for each dataset
pset_prop_rice <- seq(0.5, 1, by=0.1)^2
pset_prop_concrete <- seq(0.4, 1, by=0.1)^2


# calculate differences to exact shapley values----

# rice data
shapleys_exact_rice_rep<-bind_rows(replicate(4*length(pset_prop_rice),
                    as.data.frame(shapleys_exact_rice), simplify=F))
shapley_diff_rice <- approx_shapr_rice[, 1:ncol(
      shapleys_exact_rice)] - shapleys_exact_rice_rep
shapley_diff_rice <- bind_cols(shapley_diff_rice,
            approx_shapr_rice[, (ncol(shapley_diff_rice)+1):ncol(
              approx_shapr_rice)])
shapley_diff_rice_long <- shapley_diff_rice %>%
  pivot_longer(-c(combs_prop, runtime, approach),
               names_to = "predictor", values_to = "error") %>%
  group_by(approach, combs_prop, predictor) %>% mutate(
    mean_error=mean(error)) %>%
  ungroup()

# MAE
shapley_diff_rice_avg <- shapley_diff_rice_long %>%
  group_by(approach, combs_prop) %>%
  summarise(across(where(is.numeric), ~mean(abs(.))))


# concrete data
shapleys_exact_concrete_rep<-bind_rows(replicate(4*length(
  pset_prop_concrete),
    as.data.frame(shapleys_exact_concrete), simplify=F))
shapley_diff_concrete <- approx_shapr_concrete[, 1:ncol(
  shapleys_exact_concrete)] - shapleys_exact_concrete_rep
shapley_diff_concrete <- bind_cols(
  shapley_diff_concrete, approx_shapr_concrete[, (
    ncol(shapley_diff_concrete)+1):ncol(approx_shapr_concrete)])
shapley_diff_concrete_long <- shapley_diff_concrete %>%
  pivot_longer(-c(combs_prop, runtime, approach),
               names_to = "predictor", values_to = "error") %>%
      group_by(approach, combs_prop, predictor) %>%
  mutate(mean_error=mean(error)) %>%
     ungroup()

# MAE
shapley_diff_concrete_avg <- shapley_diff_concrete_long %>%
  group_by(approach, combs_prop) %>%
  summarise(across(where(is.numeric), ~mean(abs(.))))
```

```r
# seismic data
shapleys_exact_seismic_rep<-bind_rows(replicate(2, as.data.frame(
    shapleys_exact_seismic), simplify=F))
shapley_diff_seismic <- approx_shapr_seismic[, 1:ncol(
  shapleys_exact_seismic)] - shapleys_exact_seismic_rep
shapley_diff_seismic <- bind_cols(shapley_diff_seismic,
      approx_shapr_seismic[, (ncol(shapley_diff_seismic)+1):ncol(
        approx_shapr_seismic)])
shapley_diff_seismic_long <- shapley_diff_seismic %>% pivot_longer(
  -c(runtime, approach, combs_prop),
                  names_to = "predictor", values_to = "error") %>%
  group_by(approach, combs_prop, predictor) %>%
  mutate(mean_error=mean(error)) %>%
  ungroup()

# MAE
shapley_diff_seismic_avg <- shapley_diff_seismic_long %>%
  group_by(approach) %>%
  summarise(across(where(is.numeric), ~mean(abs(.))))



# plots to compare exact Shapley values with approximations----


# distributions of approximation errors, by predictors

# rice data
ggplot(shapley_diff_rice_long, aes(y = predictor, x = error)) +
  geom_point(position=position_dodge(width=0.5),
             aes(color=as.factor(combs_prop), group=as.factor(combs_prop)),
             size = 0.25, alpha=0.4) +
  geom_point(position=position_dodge(width=0.5),
             aes(x=mean_error,
             group=as.factor(combs_prop)), shape = 18, size = 2, color="firebrick1") +
  scale_colour_viridis_d(
    "proportion of predictor combinations sampled", direction = -1) +
  geom_vline(xintercept = 0, linetype = "dashed", color = "gray50") +
  scale_x_continuous(name =
                       "approximated Shapley value - exact Shapley value") +
  facet_wrap(~approach, nrow=4) + theme_classic() +
  theme(legend.position = "top", axis.title.y = element_blank()) +
  guides(colour = guide_legend(nrow=1,override.aes = list(size=3, alpha=1),
              direction = "horizontal", title.position = "top"))


# concrete data
ggplot(shapley_diff_concrete_long, aes(y = predictor, x = error)) +
  geom_point(position=position_dodge(width=0.5),
             aes(color=as.factor(combs_prop), group=as.factor(combs_prop)),
```

```
            size = 0.25, alpha=0.4) +
  geom_point(position=position_dodge(width=0.5),
             aes(x=mean_error,
       group=as.factor(combs_prop)), shape = 18, size = 2, color="firebrick1") +
  scale_colour_viridis_d(
    "proportion of predictor combinations sampled", direction = -1) +
  geom_vline(xintercept = 0, linetype = "dashed", color = "gray50") +
  scale_x_continuous(
    name = "approximated Shapley value - exact Shapley value") +
  facet_wrap(~approach, nrow=4) + theme_classic() +
  theme(legend.position = "top", axis.title.y = element_blank()) +
  guides(colour = guide_legend(nrow=1,override.aes = list(size=3, alpha=1),
              direction = "horizontal", title.position = "top"))

# seismic data
ggplot(shapley_diff_seismic_long, aes(y = predictor, x = error)) +
  geom_boxplot()+
  geom_point(aes(x=mean_error), position=position_dodge(width=0.5),
             shape = 18, size = 2, color="firebrick1") +
  geom_vline(xintercept = 0, linetype = "dashed", color = "gray50") +
  xlab("approximated Shapley value - exact Shapley value") +
  xlim(c(-0.06, 0.06)) +
  facet_wrap(~approach, nrow = 2) + theme_classic() +
  theme(axis.title.y = element_blank()) +
  labs(caption="Some outliers were removed for better visibility")



# MAE convergence plots-----

# rice data
ggplot(shapley_diff_rice_avg, aes(x=combs_prop, y=error, color=approach)) +
  scale_colour_manual(values = c("forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
  xlab("proportion of predictor combinations sampled") +
  ylab("mean absolute error")

# concrete data
ggplot(shapley_diff_concrete_avg, aes(x=combs_prop, y=error, color=approach)) +
  scale_colour_manual(values = c(
    "forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
  xlab("proportion of predictor combinations sampled") +
  ylab("mean absolute error")
```

```r
# ranking ability: Kendall correlation between true and approximated values----


# rice data

approaches <- c("copula", "ctree", "gaussian", "independence")
kcorr_rice <- matrix(ncol = (nrow(shapleys_exact_rice)))
for(j in 1:length(approaches)){

  apr <- approx_shapr_rice %>% filter(approach==approaches[j])

  for(i in 1:length(pset_prop_rice)){

    apr_i <- apr %>% filter(as.character(
        combs_prop)==as.character(pset_prop_rice[i]))
    kcorr <- diag(cor(t(apr_i[, 1:ncol(shapleys_exact_rice)]),
                      t(shapleys_exact_rice), method = "kendall"))
    kcorr_rice <- rbind(kcorr_rice, kcorr)
  }
}
kcorr_avg_rice <- rowMeans(na.omit(kcorr_rice))
kcorr_avg_rice <- bind_cols(na.omit(kcorr_avg_rice),
          (approx_shapr_rice %>% select(approach, combs_prop) %>% unique())) %>%
  rename(avgcorr=1)


ggplot(as.data.frame(kcorr_avg_rice), aes(x=combs_prop, y=avgcorr, color=approach)) +
  scale_colour_manual(values = c("forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
  xlab("proportion of predictor combinations sampled") +
  ylab("avg. Kendall correlation across observations")


# concrete data
kcorr_concrete <- matrix(ncol = (nrow(shapleys_exact_concrete)))
for(j in 1:length(approaches)){

  apr <- approx_shapr_concrete %>% filter(approach==approaches[j])

  for(i in 1:length(pset_prop_concrete)){

    apr_i <- apr %>% filter(as.character(combs_prop)==as.character(pset_prop_concrete[i]))
    kcorr <- diag(cor(t(apr_i[, 1:ncol(shapleys_exact_concrete)]),
                      t(shapleys_exact_concrete), method = "kendall"))
    kcorr_concrete <- rbind(kcorr_concrete, kcorr)
  }
```

```
}
kcorr_avg_concrete <- rowMeans(na.omit(kcorr_concrete))
kcorr_avg_concrete <- bind_cols(na.omit(kcorr_avg_concrete),
                   (approx_shapr_concrete %>% select(approach, combs_prop) %>% unique())) %>%
  rename(avgcorr=1)

ggplot(as.data.frame(kcorr_avg_concrete), aes(x=combs_prop, y=avgcorr, color=approach)) +
  scale_colour_manual(values = c("forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
  xlab("proportion of predictor combinations sampled") +
  ylab("avg. Kendall correlation across observations")


# seismic data (text only)
kcorr_seismic_ctree <- diag(cor(
  approx_shapr_seismic_ctree[, 1:ncol(shapleys_exact_seismic)],
  shapleys_exact_seismic, method = "kendall"))
kcorr_seismic_indep <- diag(cor(
  approx_shapr_seismic_indep[, 1:ncol(shapleys_exact_seismic)],
  shapleys_exact_seismic, method = "kendall"))
kcorr_avg_seismic_ctree <- mean(kcorr_seismic_ctree)
kcorr_avg_seismic_indep <- mean(kcorr_seismic_indep)



# (runtime plots)----

# rice data
ggplot(shapley_diff_rice_avg, aes(x=runtime, y=error, color=approach)) +
  scale_colour_manual(values = c("forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  xlim(c(0, max(shapley_diff_rice_avg$runtime)))+
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
  xlab("running time on test set (seconds)") +
  ylab("mean absolute error")


# concrete data
ggplot(shapley_diff_concrete_avg, aes(x=runtime, y=error, color=approach)) +
  scale_colour_manual(values = c("forestgreen", "blue", "black", "sienna")) +
  geom_point() + geom_line() +
  xlim(c(0, max(shapley_diff_rice_avg$runtime)))+
  geom_hline(yintercept = 0, linetype="dashed") +
  theme_classic() +
  theme(legend.position = "top")+
```

```
  xlab("running time on test set (seconds)") +
  ylab("mean absolute error")
```

Session Info

```
# R version used
sessionInfo()$R.version$version.string

## [1] "R version 4.3.2 (2023-10-31)"

# operating system
sessionInfo()$platform

## [1] "x86_64-pc-linux-gnu (64-bit)"

# packages loaded via namespace
for (package_name in sort(loadedNamespaces())) {
    print(paste(package_name, packageVersion(package_name)))
}

## [1] "backports 1.4.1"
## [1] "base 4.3.2"
## [1] "broom 1.0.5"
## [1] "cellranger 1.1.0"
## [1] "class 7.3.22"
## [1] "cli 3.6.1"
## [1] "codetools 0.2.19"
## [1] "colorspace 2.1.0"
## [1] "compiler 4.3.2"
## [1] "data.table 1.14.8"
## [1] "datasets 4.3.2"
## [1] "dials 1.2.0"
## [1] "DiceDesign 1.9"
## [1] "digest 0.6.33"
## [1] "dplyr 1.1.3"
## [1] "ellipsis 0.3.2"
## [1] "evaluate 0.22"
## [1] "fansi 1.0.5"
## [1] "farver 2.1.1"
## [1] "fastmap 1.1.1"
## [1] "forcats 1.0.0"
## [1] "foreach 1.5.2"
## [1] "furrr 0.3.1"
## [1] "future 1.33.0"
## [1] "future.apply 1.11.0"
## [1] "generics 0.1.3"
## [1] "ggplot2 3.4.4"
## [1] "globals 0.16.2"
## [1] "glue 1.6.2"
## [1] "gower 1.0.1"
## [1] "GPfit 1.0.8"
```

```
## [1] "graphics 4.3.2"
## [1] "grDevices 4.3.2"
## [1] "grid 4.3.2"
## [1] "gtable 0.3.4"
## [1] "hardhat 1.3.0"
## [1] "hms 1.1.3"
## [1] "htmltools 0.5.6.1"
## [1] "httr 1.4.7"
## [1] "infer 1.0.5"
## [1] "ipred 0.9.14"
## [1] "iterators 1.0.14"
## [1] "jsonlite 1.8.7"
## [1] "kableExtra 1.3.4"
## [1] "knitr 1.44"
## [1] "labeling 0.4.3"
## [1] "lattice 0.21.9"
## [1] "lava 1.7.2.1"
## [1] "lhs 1.1.6"
## [1] "lifecycle 1.0.3"
## [1] "listenv 0.9.0"
## [1] "lubridate 1.9.3"
## [1] "magrittr 2.0.3"
## [1] "MASS 7.3.60"
## [1] "Matrix 1.6.1.1"
## [1] "methods 4.3.2"
## [1] "modeldata 1.2.0"
## [1] "munsell 0.5.0"
## [1] "nnet 7.3.19"
## [1] "parallel 4.3.2"
## [1] "parallelly 1.36.0"
## [1] "parsnip 1.1.1"
## [1] "pillar 1.9.0"
## [1] "pkgconfig 2.0.3"
## [1] "plyr 1.8.9"
## [1] "prodlim 2023.8.28"
## [1] "progressr 0.14.0"
## [1] "purrr 1.0.2"
## [1] "R6 2.5.1"
## [1] "ranger 0.15.1"
## [1] "Rcpp 1.0.11"
## [1] "readr 2.1.4"
## [1] "readxl 1.4.3"
## [1] "recipes 1.0.8"
## [1] "reshape2 1.4.4"
## [1] "rlang 1.1.1"
## [1] "rmarkdown 2.25"
## [1] "rpart 4.1.21"
## [1] "rsample 1.2.0"
## [1] "rstudioapi 0.15.0"
## [1] "rvest 1.0.3"
```

```
## [1] "scales 1.2.1"
## [1] "shapr 0.2.3.9100"
## [1] "splines 4.3.2"
## [1] "stats 4.3.2"
## [1] "stringi 1.7.12"
## [1] "stringr 1.5.0"
## [1] "survival 3.5.7"
## [1] "svglite 2.1.2"
## [1] "systemfonts 1.0.5"
## [1] "tibble 3.2.1"
## [1] "tidymodels 1.1.1"
## [1] "tidyr 1.3.0"
## [1] "tidyselect 1.2.0"
## [1] "tidyverse 2.0.0"
## [1] "timechange 0.2.0"
## [1] "timeDate 4022.108"
## [1] "tools 4.3.2"
## [1] "tune 1.1.2"
## [1] "tzdb 0.4.0"
## [1] "utf8 1.2.3"
## [1] "utils 4.3.2"
## [1] "vctrs 0.6.4"
## [1] "viridisLite 0.4.2"
## [1] "webshot 0.5.5"
## [1] "withr 2.5.1"
## [1] "workflows 1.1.3"
## [1] "workflowsets 1.0.1"
## [1] "xfun 0.40"
## [1] "xgboost 1.7.5.1"
## [1] "xml2 1.3.5"
## [1] "yaml 2.3.7"
## [1] "yardstick 1.2.0"
```