Explaining Predictive Models with Mixed Features Using Shapley Values and Conditional Inference Trees

Annabelle Redelmeier, Martin Jullum, Kjersti Aas

Explaining predictive models with mixed features using Shapley values and conditional inference trees

Annabelle Redelmeier, Martin Jullum, and Kjersti Aas

Norwegian Computing Center, P.O. Box 114, Blindern, N-0314 Oslo, Norway
{annabelle.redelmeier,jullum,kjersti}@nr.no

Abstract. It is becoming increasingly important to explain complex, black-box machine learning models. Although there is an expanding literature on this topic, Shapley values stand out as a sound method to explain predictions from any type of machine learning model. The original development of Shapley values for prediction explanation relied on the assumption that the features being described were independent. This methodology was then extended to explain dependent features with an underlying continuous distribution. In this paper, we propose a method to explain mixed (i.e. continuous, discrete, ordinal, and categorical) dependent features by modeling the dependence structure of the features using conditional inference trees. We demonstrate our proposed method against the current industry standards in various simulation studies and find that our method often outperforms the other approaches. Finally, we apply our method to a real financial data set used in the 2018 FICO Explainable Machine Learning Challenge and show how our explanations compare to the FICO challenge Recognition Award winning team.

Keywords: Explainable AI · Shapley values · conditional inference trees · feature dependence · prediction explanation

1 Introduction

Due to the ongoing data and artificial intelligence (AI) revolution, an increasing number of crucial decisions are being made with complex automated systems. It is therefore becoming ever more important to understand how these systems make decisions. Such systems often consist of 'black-box' machine learning models which are trained to predict an outcome/decision based on various input data (i.e. features). Consider, for instance, a model that predicts the price of car insurance based on the features age and gender of the individual, type of car, time since the car was registered, and number of accidents in the last five years. For such a system to work in practice, the expert making the model, the insurance brokers communicating the model, and the policyholders vetting the model should know which features drive the price of insurance up or down.

Although there are numerous ways to explain complex models, one way is to show how the individual features contribute to the overall predicted value

for a given individual[1]. The Shapley value framework is recent methodology to calculate these contributions [15, 20, 21]. In the framework, a Shapley value is derived for each feature given a prediction (or 'black-box') model and the set of feature values for the given individual. The methodology is such that the sum of the Shapley values for the individual equals their prediction value so that the features with the largest (absolute) Shapley values are the most important.

The Shapley value concept is based on economic game theory. The original setting is as follows: Imagine a game where N players cooperate in order to maximize the *total gains* of the game. Suppose further that each player is to be given a certain payout for his or her efforts. Lloyd Shapley [19] discovered a way to distribute the total gains of the game among the players according to certain desirable axioms. For example, players that do not contribute anything get a payout of 0; two players that contribute the same regardless of other players get the same payout; and the sum of the payouts equals the total gains of the game. A player's payout is known as his or her *Shapley value*.

[15, 20, 21] translate Shapley values from the game theory setting to a machine learning setting. The cooperative game becomes the individual, the total gains of the game become the prediction value, and the players become the feature values. Then, analogous to game theory, the Shapley value of one of the features (called the *Shapley value explanation*) is how the feature contributes to the overall prediction value.

Figure 1 shows how such Shapley value explanations can be visualized for two examples of the aforementioned car insurance scenario. For the individual on the left, 'number of accidents' pulls the predicted insurance price up (its Shapley value is positive) whereas 'gender' and 'age' pull it down. The features 'type of car' and 'time since registration' only minimally affect the prediction. For the individual on the right, 'gender' and 'type of car' pull the predicted insurance price up whereas 'age', 'time since registration', and 'number of accidents' pull it marginally down. The sum of the Shapley values of each individual gives the predicted price of insurance (123.5 and 229.9 USD/month, respectively). Note that 'none' is a fixed average prediction contribution not due to any of the features in the model.

As we demonstrate in Section 2, calculating Shapley values is not necessarily straightforward or computationally simple. To simplify the estimation problem, [15, 20, 21] assume the features are independent. However, [1] shows that this may lead to severely inaccurate Shapley value estimates and, therefore, incorrect explanations when the features are not independent. [1] extends [15]'s methodology to handle dependent *continuously* distributed features by modeling/estimating the dependence between the features. However, as exemplified by the aforementioned car insurance example, practical modeling scenarios often involve a mixture of different feature types: continuous (age and time since the car was registered), discrete (number of accidents in the last five years), and categorical (gender and type of car). Thus, there is a clear need to extend the

[1] Here, 'individual' could be an individual person or an individual non-training observation - not necessarily a person.

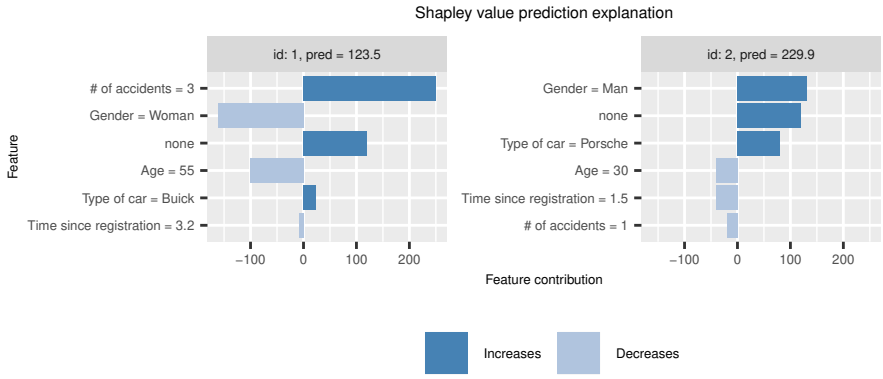

Fig. 1. An example of using Shapley values to show how the predicted price of car insurance can be broken down into the respective features.

Shapley value methodology to handle dependent mixed (i.e. continuous, discrete, ordinal, categorical) features.

While it is, in principle, possible to naively apply some of the methods proposed by [1] to discrete or categorical features, it is unlikely that they will function well. This will typically require encoding categorical features with L different categories into $L - 1$ new indicator features using one-hot encoding. The main drawback of this approach is that the feature dimension increases substantially unless there are very few categories. Computational power is already a non-trivial issue with the Shapley value framework (see [1]), so this is not a feasible approach unless the number of categories or features is very small.

The aim of this paper is to show how we can extend the Shapley value framework to handle mixed features without assuming the features are independent. We propose to use a special type of tree model, known as conditional inference trees [12], to model the dependence between the features. This is similar to [1]'s extension of [15]'s work but for mixed features. We use tree models since they are inherently good at modeling both simple and complex dependence structures in mixed data types [8]. The conditional inference tree model has the additional advantage of naturally extending to multivariate responses, which is required in this setting. Since conditional inference trees handle categorical data, this approach does not require one-hot encoding any features resulting in a much shorter computation time. In addition, we do not run the risk of estimating one-hot encoded features using a method not designed for the sort.

The rest of the paper is organized as follows. We begin by explaining the fundamentals of the Shapley value framework in an explanation setting in Section 2 and then outline how to extend the method to mixed features using conditional inference trees in Section 3. In Section 4, we present various simulation studies for both continuous and categorical features that demonstrate that our method works in a variety of settings. Finally, in Section 5, we apply our method to the

2018 FICO Explainable Machine Learning Challenge data set and show how the estimated Shapley values differ when calculated using various feature distribution assumptions. We also compare the feature importance rankings calculated using Shapley values with the rankings calculated by the 2018 FICO challenge Recognition Award winning team. In Section 6, we conclude.

The Shapley methodology from [1] is implemented in the software package `shapr` [18] in the `R` programming language [17]. Our new approach is implemented as an extension to the `shapr` package. To construct the conditional inference trees in `R`, we use the packages `party` and `partykit` [12, 13].

2 Shapley values

2.1 Shapley values in game theory

Suppose we are in a cooperative game setting with M players, $j = 1, \ldots, M$, trying to maximize a payoff. Let $\mathcal{M}$ be the set of all players and $\mathcal{S}$ any subset of $\mathcal{M}$. Then the Shapley value [19] for the jth player is defined as

$$\phi_j = \sum_{\mathcal{S} \subseteq \mathcal{M} \setminus \{j\}} \frac{|\mathcal{S}|!(M - |\mathcal{S}| - 1)!}{M!} (v(\mathcal{S} \cup \{j\}) - v(\mathcal{S})). \tag{1}$$

$v(\mathcal{S})$ is the contribution function which maps subsets of players to real numbers representing the worth or contribution of the group $\mathcal{S}$ and $|\mathcal{S}|$ is the number of players in subset $\mathcal{S}$.

In the game theory sense, each player receives ϕ_j as their payout. From the formula, we see that this payout is just a weighted sum of the player's marginal contributions to each group $\mathcal{S}$. Lloyd Shapley [19] proved that distributing the total gains of the game in this way is 'fair' in the sense that it obeys certain important axioms.

2.2 Shapley values for explainability

In a machine learning setting, imagine a scenario where we fit M features, $\boldsymbol{x} = (x_1, \ldots, x_M)$, to a univariate response y with the model $f(\boldsymbol{x})$ and want to explain the prediction $f(\boldsymbol{x})$ for a specific feature vector $\boldsymbol{x} = \boldsymbol{x}^*$. [15, 20, 21] suggest doing this with Shapley values where the predictive model replaces the cooperative game and the features replace the players. To use (1), [15] defines the contribution function $v(\mathcal{S})$ as the following expected prediction

$$v(\mathcal{S}) = \mathbb{E}[f(\boldsymbol{x}) | \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}]. \tag{2}$$

Here $\boldsymbol{x}_{\mathcal{S}}$ denotes the features in subset $\mathcal{S}$ and $\boldsymbol{x}^*_{\mathcal{S}}$ is the subset $\mathcal{S}$ of the feature vector $\boldsymbol{x}^*$ that we want to explain. Thus, $v(\mathcal{S})$ denotes the expected prediction given that the features in subset $\mathcal{S}$ take the value $\boldsymbol{x}^*_{\mathcal{S}}$.

Calculating the Shapley value for a given feature x_j thus becomes the arduous task of computing (1) but replacing $v(\mathcal{S})$ with the conditional expectation (2).

It is clear that the sum in (1) grows exponentially as the number of features, M, increases. [15] cleverly approximates this weighted sum in a method they call Kernel SHAP. Specifically, they define the Shapley values as the optimal solution to a certain weighted least squares problem. They prove that the Shapley values can explicitly be written as

$$\phi = (\boldsymbol{Z}^T \boldsymbol{W} \boldsymbol{Z})^{-1} \boldsymbol{Z}^T \boldsymbol{W} \boldsymbol{v}, \tag{3}$$

where $\boldsymbol{Z}$ is the $2^M \times (M+1)$ binary matrix representing all possible combinations of the M features, $\boldsymbol{W}$ is the $2^M \times 2^M$ diagonal matrix containing Shapley weights, and $\boldsymbol{v}$ is the vector containing $v(\mathcal{S})$ for every $\mathcal{S}$. The full derivation is described in [1].

To calculate (3), we still need to compute the contribution function $v(\mathcal{S})$ for different subsets of features, $\mathcal{S}$. When the features are continuous, we can write the conditional expectation (2) as

$$\mathbb{E}[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] = \mathbb{E}[f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] = \int f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}^*_{\mathcal{S}}) p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})\, d\boldsymbol{x}^*_{\bar{\mathcal{S}}}, \tag{4}$$

where $\boldsymbol{x}_{\bar{\mathcal{S}}}$ is the vector of features not in $\mathcal{S}$ and $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$ is the conditional distribution of $\boldsymbol{x}_{\bar{\mathcal{S}}}$ given $\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}$. Note that in the rest of the paper we use $p(\cdot)$ to refer to both probability mass functions and density functions (made clear by the context). We also use lower case x-s for both random variables and realizations to keep the notation concise.

Since the conditional probability function is rarely known, [15] replaces it with the simple (unconditional) probability function

$$p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}) = p(\boldsymbol{x}_{\bar{\mathcal{S}}}). \tag{5}$$

The integral then becomes

$$\mathbb{E}[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] = \int f(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}^*_{\mathcal{S}}) p(\boldsymbol{x}_{\bar{\mathcal{S}}})\, d\boldsymbol{x}^*_{\bar{\mathcal{S}}}, \tag{6}$$

which is estimated by randomly drawing K times from the full training data set and calculating

$$v_{\text{KerSHAP}}(\mathcal{S}) = \frac{1}{K} \sum_{k=1}^{K} f(\boldsymbol{x}^k_{\bar{\mathcal{S}}}, \boldsymbol{x}^*_{\mathcal{S}}), \tag{7}$$

where $\boldsymbol{x}^k_{\bar{\mathcal{S}}}$, $k = 1, \ldots, K$ are the samples from the training set and $f(\cdot)$ is the estimated prediction model.

Unfortunately, when the features are not independent, [1] demonstrates that naively replacing the conditional probability function with the unconditional one leads to very inaccurate Shapley values. [1] then proposes multiple methods for estimating $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}})$ without relying on the naive assumption in (5). However, these methods are only constructed for continuous features. In the next section we demonstrate how we can use conditional inference trees to extend the current Shapley framework to handle mixed features.

3 Extending the Shapley framework with conditional inference trees

Conditional inference trees (ctree) [12] is a type of recursive partitioning algorithm like CART (classification and regression trees) [4] and C4.5 [16]. Just like these algorithms, ctree builds trees recursively, making binary splits on the feature space until a given stopping criterion is fulfilled. The difference between ctree and CART/C4.5 is how the feature/split point and stopping criterion are chosen. CART and C4.5 solve for the feature and split point simultaneously: each feature and split point is tried together and the best pair is the combination that results in the smallest error (often based on the squared error loss or binary cross-entropy loss depending on the response). Ctree, on the other hand, proceeds sequentially: the splitting feature is chosen using statistical significance tests and then the split point is chosen using any type of splitting criterion [12]. According to [12], choosing the splitting feature without first checking for the potential split points avoids being biased towards features with many split points. In addition, unlike CART and C4.5, ctree is defined independently of the dimension of the response variable. This is advantageous since proper handling of multivariate responses is crucial for our problem.

3.1 Conditional inference tree algorithm

Suppose that we have a training data set with p features, a q dimensional response, and n observations: $\{\boldsymbol{y}_i, \boldsymbol{x}_i\}_{i=1,\dots,n}$ with $\boldsymbol{y}_i = (y_{i1}, \dots y_{iq})$ and $\boldsymbol{x}_i = (x_{i1}, \dots, x_{ip})$. Suppose further that the responses come from a sample space $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_q$ and the features come from a sample space $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$. Then conditional inference trees are built using the following algorithm:

1. For a given node in the tree, test the global null hypothesis of independence between all of the p features and the response $\boldsymbol{y}$. If the global hypothesis cannot be rejected, do not split the node. Otherwise, select the feature x_j that is the least independent of $\boldsymbol{y}$.
2. Choose a splitting point in order to split $\mathcal{X}_j$ into two disjoint groups.

Steps 1 and 2 are repeated until no nodes are split.

The global null hypothesis can be written as

$$H_0 : \cap_{j=1}^{p} H_0^j,$$

where the p partial hypotheses are

$$H_0^j : F(\boldsymbol{Y}|X_j) = F(\boldsymbol{Y}),$$

and $F(\cdot)$ is the distribution of $\boldsymbol{Y}$.

Specifically, we calculate the p P-values for the partial hypotheses and combine them to form the global null hypothesis P-value. If the P-value for the global null hypothesis is smaller than some predetermined level α, we reject the

global null hypothesis and assume that there is some dependence between the features and the response. The feature that is the least independent of the response (i.e. has the smallest partial P-value) becomes the splitting feature. If the global null hypothesis is not rejected, we do not split the node. The size of the tree is controlled using the parameter α. As α increases, we are more likely to reject the global null hypothesis and therefore split the node. This results in deeper trees. However, if α is too large, we risk that the tree overfits the data.

Step 2 can be done using any type of splitting criterion, specifically it can be done with the permutation test framework devised by [12]. Note that this method is not tied to a specific feature type and can be used with mixtures of continuous, discrete, ordinal, and categorical features. We refer the reader to the original paper [12] for more details on how to form the test statistic, associated distribution, and P-values.

3.2 Extending the Shapley value framework with conditional inference trees

As already mentioned, one of the main limitations with the Shapley value framework is estimating the contribution function (2) when the conditional distribution of the features is unknown but the features are assumed dependent. [1] estimates (2) by modeling the conditional probability density function

$$p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}) \tag{8}$$

using various approaches. Then, [1] samples K times from this modeled conditional distribution function and uses these samples to estimate the integral (4) using (7).

We extend this approach to mixed features by modeling the conditional distribution function (8) using conditional inference trees. We fit a tree to our training data where the features are $\boldsymbol{x}_{\mathcal{S}}$ and the response is $\boldsymbol{x}_{\bar{\mathcal{S}}}$ with the algorithm described in Section 3.1. Then for a given $\boldsymbol{x}^*_{\mathcal{S}}$, we find its leaf in the tree and sample K times from the $\boldsymbol{x}_{\bar{\mathcal{S}}}$ part of the training observations in that node to obtain $\boldsymbol{x}^k_{\bar{\mathcal{S}}}$, $k = 1, \ldots, K$. Finally, we use these samples to estimate (4) using the approximation (7).

We fit a new tree to every combination of features $\boldsymbol{x}_{\mathcal{S}}$ and response $\boldsymbol{x}_{\bar{\mathcal{S}}}$. Once $v(\mathcal{S})$ is estimated for every $\mathcal{S}$, we follow [14]'s steps and estimate the Shapley value of this feature with (3). Since conditional inference trees handle continuous, discrete, ordinal, and categorical features; univariate and multivariate responses; and any type of dependence structure, using conditional inference trees to estimate (8) is a natural extension to [1]'s work. Below, we use the term *ctree* to refer to estimating Shapley value explanations using conditional inference trees.

4 Simulation studies

In this section, we discuss two simulation studies designed to compare different ways to estimate Shapley values. Specifically, we compare our ctree estimation

approach with [15]'s independence estimation approach (below called *independence*) and [1]'s empirical and Gaussian estimation approaches. A short description of each approach is in Table 1.

Method	Citation	Description
independence	[14]	Assume the features are independent. Assume (2) is (6) and estimate it with (7) where $x_{\bar{S}}^k$ are sub-samples from the training data set.
empirical	[1]	Calculate the distance between the set of features being explained and every training instance. Use this distance to calculate a weight for each training instance. Approximate (2) using a function of these weights.
Gaussian (100)	[1]	Assume the features are jointly Gaussian. Estimate the mean/covariance of this conditional distribution and then sample 100 times from this distribution. Estimate (2) with (7) using this sample.
Gaussian (1000)	[1]	The same as Gaussian (100), but we sample 1000 times.
ctree		See Section 3.2. Set $\alpha = 0.5$.
ctree-onehot		Convert the categorical features into one-hot encoded features and then apply the algorithm in Section 3.2 to these binary features. This approach is used only as a reference.

Table 1. A short description of the approaches used to estimate (8) in the simulation studies.

The independence, empirical, and Gaussian approaches are all implemented in the `R` package `shapr` [18]. We implement the ctree method in the `shapr` package as an additional method. Building the conditional inference trees for each combination of features is done using either the `party` package or `partykit` package in `R` [12, 13]. Although `party` is faster than `partykit`, it sometimes runs into a convergence error related to the underlying linear algebra library in `R` (error code 1 from Lapack routine 'dgesdd'). We therefore fall back to `partykit` when this error occurs. Both packages typically give identical results.

In the first simulation study, we simulate only categorical features and in the second, we simulate both categorical and continuous features. Then, we estimate the Shapley values of each test observation with the methods in Table 1 and compare them against the truth using a mean absolute error type of performance measure.

For simplicity, in both situations we restrict ourselves to a linear predictive function of the form

$$f(\boldsymbol{x}) = \alpha + \sum_{\{j:j\in\mathcal{C}_{\text{cat}}\}} \sum_{l=2}^{L} \beta_{jl}\mathbf{1}(x_j = l) \quad + \sum_{\{j:j\in\mathcal{C}_{\text{cont}}\}} \gamma_j x_j, \tag{9}$$

where $\mathcal{C}_{\text{cat}}$ and $\mathcal{C}_{\text{cont}}$ denote, respectively, the set of categorical and continuous features, L is the number of categories for each of the categorical features, and $\mathbf{1}(x_j = l)$ is the indicator function taking the value 1 if $x_j = l$ and 0 otherwise. α, β_{jl} for $j \in \mathcal{C}_{\text{cat}}$, $l = 2, \ldots, L$, and γ_j for $j \in \mathcal{C}_{\text{cont}}$ are the parameters in the linear model. We define $M = |\mathcal{C}_{\text{cat}}| + |\mathcal{C}_{\text{cont}}|$, where $|\mathcal{C}_{\text{cat}}|$ and $|\mathcal{C}_{\text{cont}}|$ denote the number of categorical and continuous features, respectively.

The empirical and Gaussian methods cannot handle categorical features. For these methods, we transform the categorical features into one-hot encoded features. If the categorical feature originally has L categories, the one-hot encoded transformation creates $L-1$ binary features representing the second, third (etc) categories. The first category is represented by the intercept. The Shapley value of each categorical feature is then the sum of the Shapley values of the corresponding one-hot encoded features.

4.1 Evaluation method

We measure the performance of each method based on the mean absolute error (MAE), across both the features and sample space. This is defined as

$$\text{MAE(method } q) = \frac{1}{M} \sum_{j=1}^{M} \sum_{i=1}^{T} p(\boldsymbol{x}_i) |\phi_{j,\text{true}}(\boldsymbol{x}_i) - \phi_{j,q}(\boldsymbol{x}_i)|, \tag{10}$$

where $\phi_{j,q}(\boldsymbol{x})$ and $\phi_{j,\text{true}}(\boldsymbol{x})$ denote, respectively, the Shapley value estimated with method q and the corresponding true Shapley value for the prediction $f(\boldsymbol{x})$. In addition, M is the number of features and T is the number of test observations. For the case with only categorical features, the set $\{\boldsymbol{x}_i : i = 1, \ldots, T\}$ corresponds to all the unique combinations of features and $p(\boldsymbol{x}_i)$ is the probability mass function of $\boldsymbol{x}$ evaluated at $\boldsymbol{x}_i$[2]. In the case where we have both categorical and numerical features, the set $\{\boldsymbol{x}_i : i = 1, \ldots, T\}$ is sampled from the distribution of $\boldsymbol{x}$ and $p(\boldsymbol{x}_i)$ is set to $1/T$ for all i.

4.2 Simulating dependent categorical features

To simulate M dependent categorical features with L categories each, we first simulate an M-dimensional Gaussian random variable with a specified mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

$$(\tilde{x}_1, \ldots, \tilde{x}_M) \sim N_M(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{11}$$

We then transform each feature, $\tilde{x}_j$, into a categorical feature, x_j, using the following transformation:

$$x_j = l, \text{ if } v_l < \tilde{x}_j \leq v_{l+1}, \text{ for } l = 1, \ldots, L \text{ and } j = 1, \ldots, M, \tag{12}$$

[2] If there are many categorical features or number of categories, we instead use a subset of the most likely combinations and scale the probabilities such that they sum to 1 over those combinations.

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

For the simulation setting in Section 4.5 where there are both categorical and continuous features, we first sample $M = |\mathcal{C}_{\text{cat}}| + |\mathcal{C}_{\text{cont}}|$ features using (11). Then we transform the features $\tilde{x}_j$ where $j \in \mathcal{C}_{\text{cat}}$ to categorical ones using (12), and leave the remaining features untouched (i.e letting $x_j = \tilde{x}_j$, when $j \in \mathcal{C}_{\text{cont}}$). This imposes dependence both within and between all feature types.

4.3 Calculating the true Shapley values

To evaluate the performance of the different methods with the MAE from Section 4.1, we need to calculate the true Shapley values, $\phi_{j,\text{true}}(\boldsymbol{x}^*)$, $j = 1, \ldots, M$, for all feature vectors where $\boldsymbol{x}^* = \boldsymbol{x}_i$, $i = 1, \ldots, T$. This requires the true conditional expectation (2) for all feature subsets $\mathcal{S}$. We compute these expectations differently depending on whether the features are all categorical or whether there are both categorical and continuous features. The linearity of the predictive function (9) helps to simplify the computations for the latter case. Since there is no need of it in the former case, we present that case more generally.

When the features are all categorical, the desired conditional expectation can be written as

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

where $\mathcal{X}_{\bar{\mathcal{S}}}$ denotes the feature space of the feature vector $\boldsymbol{x}_{\bar{\mathcal{S}}}$ which contains $|\bar{\mathcal{S}}|^L$ unique feature combinations. Thus, all we need is the conditional probability $p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}_{\mathcal{S}}^*)$ for each combination of $\boldsymbol{x}_{\bar{\mathcal{S}}} \in \mathcal{X}_{\bar{\mathcal{S}}}$. Using standard probability theory, this conditional probability can be written as

$$p(\boldsymbol{x}_{\bar{\mathcal{S}}}|\boldsymbol{x}_{\mathcal{S}}) = \frac{p(\boldsymbol{x}_{\bar{\mathcal{S}}}, \boldsymbol{x}_{\mathcal{S}})}{p(\boldsymbol{x}_{\mathcal{S}})},$$

and then evaluated at the desired $\boldsymbol{x}_{\mathcal{S}}^*$. Since all feature combinations correspond to hyperrectangular subspaces of Gaussian features, we can compute all joint probabilities exactly using the cut-offs $v_1, \ldots, v_{L+1}$:

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

for $l_j = 1, \ldots, L$, $j = 1, \ldots, M$. Here $p(\cdot)$ denotes the joint probability mass function of $\boldsymbol{x}$ while $P(\cdot)$ denotes the joint continuous distribution function of $\tilde{\boldsymbol{x}}$. The probability on the right is easy to compute based on the cumulative distribution function of the multivariate Gaussian distribution (we used the `R` package `mvtnorm` [11]). The marginal and joint probability functions based on

only a subset of the features are computed analogously based on a subset of the full Gaussian distribution, which is also Gaussian.

For the situation where some of the features are categorical and some are continuous, the computation of the conditional expectation is more arduous. However, due to the linearity of the predictive function (9), the conditional expectation reduces to a linear combination of two types of univariate expectations. Let $\mathcal{S}_{\text{cat}}$ and $\bar{\mathcal{S}}_{\text{cat}}$ refer to, respectively, the $\mathcal{S}$ and $\bar{\mathcal{S}}$ part of the categorical features, $\mathcal{C}_{\text{cat}}$, with analogous sets $\mathcal{S}_{\text{cont}}$ and $\bar{\mathcal{S}}_{\text{cont}}$ for the continuous features. We then write the desired conditional expectation as

$$\begin{aligned}
\mathbb{E}[f(\boldsymbol{x})|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] &= \mathbb{E}\left[\alpha + \sum_{j\in\mathcal{C}_{\text{cat}}}\sum_{l=2}^{L}\beta_{jl}\mathbf{1}(x_j = l) + \sum_{j\in\mathcal{C}_{\text{cont}}}\gamma_k x_j \,\middle|\, \boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}\right] \\
&= \alpha + \sum_{j\in\mathcal{C}_{\text{cat}}}\sum_{l=2}^{L}\beta_{jl}\mathbb{E}[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] + \sum_{j\in\mathcal{C}_{\text{cont}}}\gamma_j\mathbb{E}[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] \\
&= \alpha + \sum_{j\in\bar{\mathcal{S}}_{\text{cat}}}\sum_{l=2}^{L}\beta_{jl}\mathbb{E}[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] + \sum_{j\in\mathcal{S}_{\text{cat}}}\sum_{l=2}^{L}\beta_{jl}\mathbf{1}(x^*_j = l) \\
&+ \sum_{j\in\bar{\mathcal{S}}_{\text{cont}}}\gamma_j\mathbb{E}[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}] + \sum_{j\in\mathcal{S}_{\text{cont}}}\gamma_j x^*_j.
\end{aligned}$$

Then, we just need expressions for the two conditional expectations: $\mathbb{E}[\mathbf{1}(x_j = l)|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}]$ for $j \in \bar{\mathcal{S}}_{\text{cat}}$ and $\mathbb{E}[x_j|\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{x}^*_{\mathcal{S}}]$ for $j \in \bar{\mathcal{S}}_{\text{cont}}$. To calculate them, we use results from [3] on selection (Gaussian) distributions in addition to basic probability theory and numerical integration. Specifically, the conditional expectation for the continuous features takes the form

$$\mathbb{E}[x_j|\boldsymbol{x}_{\mathcal{S}}] = \int_{-\infty}^{\infty} x g(x)\frac{p(\boldsymbol{x}_{\mathcal{S}}|x_j = x)}{p(\boldsymbol{x}_{\mathcal{S}})}\,dx, \tag{13}$$

where $g(x)$ denotes the density of the standard normal (Gaussian) distribution, $p(\boldsymbol{x}_{\mathcal{S}}|x_j = x)$ is the conditional distribution of $\boldsymbol{x}_{\mathcal{S}}$ given x_j, and $p(\boldsymbol{x}_{\mathcal{S}})$ is the marginal distribution of $\boldsymbol{x}_{\mathcal{S}}$. The latter two are both Gaussian and can be evaluated at the specific vector $\boldsymbol{x}^*_{\mathcal{S}}$ using the `R` package `mvtnorm` [11]. Finally, the integral is solved using numerical integration.

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

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

which can be evaluated similarly to (13) and solved with numerical integration. Once we have computed the necessary conditional expectations for each of the

2^M feature subsets $\mathcal{S}$, we compute the Shapley values using (3). This goes for both the pure categorical case and the case with both categorical and continuous features.

4.4 Simulation study with only categorical features

We evaluate the performance of the different Shapley value approximation methods in the case of only categorical features with six different experimental setups. Table 2 describes these different experiments.

In each experiment, we sample $n_{\text{train}} = 1000$ training observations using the approach from Section 4.2, where the mean $\boldsymbol{\mu}$ is $\mathbf{0}$ and the covariance matrix is constructed with $\Sigma_{j,j} = 1$, $j = 1, \dots, M$, $\Sigma_{i,j} = \rho$, for $i \neq j$, where $\rho \in \{0, 0.1, 0.3, 0.5, 0.8, 0.9\}$. We set the response to

$$y_i = \alpha + \sum_{j=1}^{M} \sum_{l=2}^{L} \beta_{jl} \mathbf{1}(x_{ij} = l) + \varepsilon_i, \tag{14}$$

where x_{ij} is the jth feature of the ith training observation, ε_i, $i = 1, \dots, n_{\text{train}}$, are i.i.d. random variables sampled from the distribution $N(0, 0.01)$, and α, β_{jl}, $j = 1, \dots, M$, $l = 1, \dots, L$ are parameters sampled from $N(0, 1)$, which are fixed for every experiment. The predictive model, $f(\cdot)$, takes the same form without the noise term (i.e. (9) with $\mathcal{C}_{\text{cont}} = \emptyset$), where the parameters are fit to the n_{train} training observations using standard linear regression. Then, we estimate the Shapley values using the different methods from Table 1.

Table 2 shows that only ctree and the independence method are used when $M > 4$. This is because for $M > 4$, the methods that require one-hot encoding are too computationally expensive. In the same three cases, the number of unique feature combinations (M^L) is so large that a subset of the $T = 2000$ most likely feature combinations are used instead – see the discussion related to (10).

M	L	T	**Categorical cut-off values**	**Methods used**
3	3	27	$(-\infty, 0, 1, \infty)$	all
3	4	81	$(-\infty, -0.5, 0, 1, \infty)$	all
4	3	64	$(-\infty, 0, 1, \infty)$	all
5	6	2000	$(-\infty, -0.5, -0.25, 0, 0.9, 1, \infty)$	ctree, independence
7	5	2000	$(-\infty, -0.5, -0.25, 0, 1, \infty)$	ctree, independence
10	4	2000	$(-\infty, -0.5, 0, 1, \infty)$	ctree, independence

Table 2. An outline of the simulation study when using only categorical features. M denotes the number of features, L denotes the number of categories, and T denotes the number of unique test observations used to compute (10).

The results of these experiments are shown in Table 3. When the dependence between the features is small, the performance of each method is almost the same.

Note that when the correlation, ρ, is 0 (i.e. the features are independent) and $M \leq 4$, the ctree and independence methods perform equally in terms of MAE, and, in fact, give identical Shapley values. This is because when $\rho = 0$, ctree never rejects the hypothesis of independence when fitting any of the trees. As a result, ctree weighs all training observations equally which is analogous to the independence method. This ability to adapt the complexity of the dependence modeling to the actual dependence in the data is a major advantage of the ctree approach. When $M > 4$, the results of the independence and ctree methods for $\rho = 0$ are slightly different. The reason is that when the dimension is large, ctree tests many more hypotheses and therefore is more likely to reject some of the hypotheses. Since the independence method performs better than ctree in these cases, this suggests that the parameter α could be reduced for higher dimensions to improve the performance in low-correlation settings. This remains to be investigated, however.

As expected, the ctree method outperforms the independence method unless the dependence between the features is very small. The ctree approach also always outperforms (albeit marginally) the empirical, Gaussian, and ctree-onehot approaches. In addition, a major advantage of using ctree is that it does not require one-hot encoding. Since the computational complexity of computing Shapley values grows exponentially in the number of features (one-hot encoded or not), the computation time for methods requiring one-hot-encoding grows quickly compared to ctree. In Table 4, we show the average run time (in seconds) per test observation of each method. The average is taken over all correlations since the computation times are almost the same for each correlation.

The empirical method is the fastest amongst the one-hot encoded methods and is still between two and five times slower than the ctree method. This means that if the number of features/categories is large, using one-hot encoding is not suitable. For the Gaussian method we calculate (7) using both 100 and 1000 samples from the conditional distribution. Table 3 shows that the MAE is slightly smaller in the latter case but from Table 4, we see that it is nearly three times slower. Such a small performance increase is probably not worth the extra computation time.

4.5 Simulation study with both categorical and continuous features

We also perform a simulation study with both categorical and continuous features. Because we need to use numerical integration to calculate the true Shapley values (see Section 4.3), the computational complexity is large even for lower-dimensional settings. Therefore, we restrict ourselves to an experiment with two categorical features with $L = 4$ categories each and two continuous features. Unless otherwise mentioned, the simulation setup follows that of Section 4.4. As described in Section 4.2, we simulate dependent categorical/continuous data by only transforming two of the four original Gaussian features. The cut-off vector for the categorical features is set to $(-\infty, -0.5, 0, 1, \infty)$. Similarly to (14), the

M	L	**Method**	ρ					
			0	0.1	0.3	0.5	0.8	0.9
3	3	empirical	0.0308	0.0277	0.0358	0.0372	0.0419	0.0430
		Gaussian (100)	0.0308	0.0237	0.0355	0.0330	0.0320	0.0384
		Gaussian (1000)	0.0307	0.0236	0.0354	0.0327	0.0318	0.0383
		ctree-onehot	0.0278	0.0196	0.0345	0.0363	0.0431	0.0432
		ctree	**0.0274**	**0.0191**	**0.0302**	**0.0310**	**0.0244**	**0.0259**
		independence	**0.0274**	**0.0191**	0.0482	0.0777	0.1546	0.2062
3	4	empirical	0.0491	0.0465	0.0447	0.0639	0.0792	0.0659
		Gaussian (100)	0.0402	0.0350	0.0358	0.0620	0.0762	0.0724
		Gaussian (1000)	0.0403	0.0353	0.0361	0.0624	0.0763	0.0738
		ctree-onehot	0.0324	0.0244	0.0429	0.0617	0.0808	0.0680
		ctree	**0.0318**	0.0331	**0.0369**	**0.0422**	**0.0416**	**0.0291**
		independence	**0.0318**	**0.0283**	0.0774	0.1244	0.2060	0.2519
4	3	empirical	0.0385	0.0474	0.0408	0.0502	0.0473	0.0389
		Gaussian (100)	0.0312	0.0381	0.0327	0.0459	0.0475	0.0409
		Gaussian (1000)	0.0312	0.0385	0.0330	0.0453	0.0480	0.0410
		ctree-onehot	0.0234	0.0305	0.0402	0.0530	0.0484	0.0397
		ctree	**0.0223**	0.0414	**0.0387**	**0.0453**	**0.0329**	**0.0253**
		independence	**0.0223**	**0.0355**	0.0961	0.1515	0.2460	0.2848
5	6	ctree	0.0237	0.0492	**0.0621**	**0.0760**	**0.0767**	**0.0899**
		independence	**0.0222**	**0.0469**	0.1231	0.1803	0.2835	0.3039
7	5	ctree	0.0209	**0.0333**	**0.0402**	**0.0542**	**0.0530**	**0.0559**
		independence	**0.0193**	0.0345	0.0794	0.1294	0.1908	0.2397
10	4	ctree	0.0169	**0.0505**	**0.0617**	**0.0607**	**0.0627**	**0.0706**
		independence	**0.0153**	0.0544	0.1593	0.2180	0.3017	0.3412

Table 3. The MAE of each method and correlation, ρ, for each experiment. The bolded numbers denote the smallest MAE per experiment and ρ.

response is given by

$$y_i = \alpha + \sum_{j=1}^{2}\sum_{l=2}^{4} \beta_{jl}\mathbf{1}(x_{ij} = l) + \sum_{j=3}^{4} \gamma_j x_{ij} + \varepsilon_i,$$

for which we fit a linear regression model of the same form without the error term to act as the predictive model $f(\cdot)$.

Then, we estimate the Shapley values using the methods from Table 1 except for the Gaussian method with 1000 samples. This method is excluded since Section 4.4 showed that its performance was very similar to that of the Gaussian method with 100 samples but significantly more time consuming. To compare the performance of the different methods, we sample $T = 500$ observations from the joint distribution of the features and compute the MAE using (10) as described in Section 4.1.

The results are displayed in Table 5. The Gaussian method is the best performing method when $\rho = 0.1, 0.3, 0.5$ while the empirical and ctree methods are the best performing when $\rho = 0.8$ and $\rho = 0.9$, respectively. The results are not

M	L	T	**Method**	**Mean time per test obs**
3	3	27	empirical	0.086
			Gaussian (100)	4.833
			Gaussian (1000)	13.295
			ctree-onehot	0.338
			ctree	0.040
			independence	0.013
3	4	64	empirical	0.553
			Gaussian (100)	8.041
			Gaussian (1000)	29.160
			ctree-onehot	1.807
			ctree	0.023
			independence	0.007
4	3	81	empirical	0.293
			Gaussian (100)	3.845
			Gaussian (1000)	12.983
			ctree-onehot	0.841
			ctree	0.052
			independence	0.012
5	6	2000	ctree	0.118
			independence	0.030
7	5	2000	ctree	0.590
			independence	0.158
10	4	2000	ctree	6.718
			independence	2.066

Table 4. The mean run time (in seconds) per test observation, T, where the mean is taken over all correlations, ρ.

surprising since we only have two categorical features. With more categorical features or categories, we expect that the ctree method would outperform the other ones when ρ is not small. We also show the run time of each method in Table 6. The one-hot encoded methods are between nine and 75 times slower than the ctree method. This demonstrates, again, the value of using the ctree method when estimating Shapley values with categorical features.

5 Real data example

Although there is a growing literature of how to explain black-box models, there are very few studies that focus on quantifying the relevance of these methods [2]. This makes it difficult to compare different explainability methods on a real data set since there is no ground truth. One partial solution is to compare how different explainability models rank the same features for predictions based on specific test observations.

M	L	**Method**	ρ 0	0.1	0.3	0.5	0.8	0.9
2 cont/2 cat	4	empirical	0.0853	0.0852	0.0898	0.0913	**0.0973**	0.1027
		Gaussian (100)	0.0570	**0.0586**	**0.0664**	**0.0662**	0.1544	0.2417
		ctree-onehot	0.0266	0.0714	0.1061	0.1024	0.1221	0.1188
		ctree	**0.0093**	0.0848	0.1073	0.1060	0.0977	**0.0917**
		independence	**0.0093**	0.0790	0.2178	0.3520	0.5524	0.6505

Table 5. The MAE of each method and correlation, ρ, for the experiment with two continuous and two categorical features ($L = 4$ categories each). The bolded numbers denote the smallest MAE per ρ.

M	L	T	**Method**	**Mean time per test obs**
4	4	500	empirical	0.758
			Gaussian (100)	5.914
			ctree-onehot	1.514
			ctree	0.082
			independence	0.057

Table 6. The mean run time (in seconds) per test observation, T, where the mean is taken over all correlations, ρ. The simulation study has two continuous features and two categorical features ($L = 4$ categories each).

In this section, we use a data set from the 2018 FICO Explainable Machine Learning Challenge [9] aimed at motivating the creation of explainable predictive models. The data set is of Home Equity Line of Credit (HELOC) applications made by homeowners. The response is a binary feature called RiskPerformance that takes the value 1 ('Bad') if the customer is more than 90 days late on his or her payment and 0 ('Good') otherwise. 52 percent of customers have the response 'Bad' and 48 percent have the response 'Good'. There are 23 features: 21 continuous and two categorical (with eight and nine categories, respectively) which can be used to model the probability of being a 'Bad' customer. Features with the value -9 are assumed missing. We remove the rows where all features are missing.

We first use this data set to compare the Shapley values calculated using the independence approach with those calculated with our ctree approach. Then, for a few test observations, we see how the Shapley explanations compare with the explanations from the 2018 FICO challenge Recognition Award winning team from Duke University [5] (hereafter referred to as just 'Duke').

After removing the missing data and a test set of 100 observations, we use the remaining 9,765 observations to train a 5-fold cross validation (CV) model using xgboost [6] and then average these to form the final model. Our model achieves an accuracy of 0.737 (compared to Duke's accuracy of 0.74). In our experience, explanation methods often behave differently when there is dependence

between the features. As a measure of dependence, we use the standard Pearson correlation for all continuous features, Cramer's V correlation measure [7] for categorical features, and the correlation ratio [10] for continuous and categorical features. The feature 'MSinceMostRecentInqexcl7days' is the least correlated with the rest of the features with correlations between -0.109 and 0.07. The 22 other features are strongly correlated with at least one other feature (max absolute correlations between 0.4 and 0.99).

Turning to the Shapley value comparisons, we estimate the Shapley values of the features belonging to the 100 test observations using the independence approach and the ctree approach. Then, we plot the Shapley value estimates against each other for a selection of four features in Figure 2. The top left panel shows the Shapley values of one of the two categorical features ('MaxDelq2PublicRecLast12M'). Both methods give Shapley values fairly close to 0 for this feature, but there are some differences. The top right panel shows an example of a feature ('ExternalRiskEstimate') where the two methods estimate quite different Shapley values. Since these are some of the largest (absolute) Shapley values, this is one of the most influential features. We also see that for most test observations, the independence method estimates more extreme Shapley values than the ctree method.

The bottom left panel shows a feature ('MSinceMostRecentInqexcl7days') where the two methods estimate relatively similar Shapley values. As noted above, this feature is the least correlated with the rest of the features. We believe the two methods behave similarly because the independence method performs best when dealing with nearly independent features. Finally, the bottom right panel shows a feature ('NumTrades90Ever2DerogPubRec') where the independence method assigns most test observations a Shapley value very close to 0 while the ctree method does not. Although not plotted, we see this trend for 6 out of the 23 features. We notice that each of these 6 features are highly correlated with at least one other feature (max absolute correlation between 0.46 and 0.99). We suspect that the methods behave differently because of the independence method's failure to account for dependence between features. We also colour three random test observations to show that for some test observations (say 'green'), all Shapley values are estimated very similarly for the two methods, while for others (say 'blue'), the methods are sometimes quite different.

We also attempt to compare explanations based on Shapley values (estimated with either the independence or ctree method) with those based on Duke's approach [5] We reinforce that there is *no ground truth* when it comes to explanations and that Duke does not use Shapley values in their solution. Therefore, we only compare how these three methods *rank* feature importance for specific test observations.

Duke's explainability approach is based on 10 smaller regression models fit to 10 partitions (below called 'groups') of the features. They use a combination of learned weights and risks to calculate the most influential group for each customer/test observation. Based on our understanding, if the customer has a large predicted probability of being 'Bad', the largest weight $\times$ risk is the

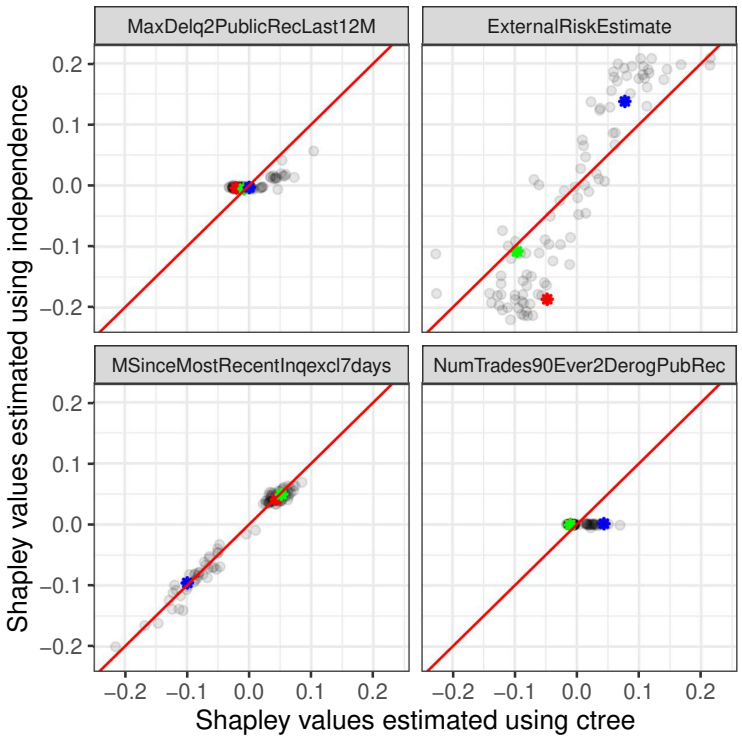

Fig. 2. The Shapley values of 100 test observations calculated using both the independence and the ctree method for four of the 23 features.

most influential group (given rank 1), while for small predicted probabilities, the largest weight divided by risk is given rank 1. It is not clear how they rank medium-range predictions.

We speculate that Duke does not properly account for feature dependence since they fit 10 independent models that only interact using an overall learned risk for each model. To test this hypothesis, we compare the group rankings of a few[3] customers/test observations calculated by 1. Duke, 2. The Shapley approach under the independence assumption, and 3. Our new Shapley approach that uses conditional inference trees.

To calculate group importance based on Shapley values for a given test observation, we first estimate the Shapley values of each feature either either the independence or ctree approach. Then, we sum the Shapley values of the features belonging to the same group. This gives 10 new grouped Shapley values. Finally, we rank the grouped Shapley values by giving rank 1 to the group with

[3] Duke's explanations were not readily available. For a given test observation, we had to manually input 23 feature values into a web application to get an explanation. Therefore, it was too time consuming to compare many test observations.

the largest absolute grouped Shapley value. While the prediction models being compared here are different (we use an xgboost model while Duke does not), the two models have very similar overall performance and give similar predictions to specific test observations. We believe this validates the rough comparison below.

We observe that for test observations with a large predicted probability of being 'Bad', there is little pattern among the rankings calculated by the three explanation methods. However, when Duke and independence give a group the same ranking out of 10 (and ctree does not), we notice that this group includes at least one feature that is very correlated with a feature in another group. On the other hand, for test observations with a small predicted probability of being 'Bad', we notice that all three explanation methods rank the groups similarly. Again, the only time ctree ranks a group differently than Duke and independence (but Duke and independence rank similarly), the group includes at least one feature highly correlated with a feature in another group.

6 Conclusion

The aim of this paper was to extend [14] and [1]'s Shapley methodology to explain mixed dependent features using conditional inference trees [12]. We showed in two simulation studies that when the features are even mildly dependent, it is advantageous to use our ctree method over the traditional independence method. Although ctree often has comparable accuracy to some of [1]'s methods, those methods require transforming the categorical features to one-hot encoded features. We demonstrated that such one-hot encoding leads to a substantial increase in computation time, making it infeasible in high dimensions.

We also demonstrated our methodology on a real financial data set. We first compared the Shapley values of 100 test observations calculated using the independence and ctree approaches. We noticed that the methods performed similarly for an almost independent feature but otherwise performed quite differently.

Then, we compared explanations based on the independence/ctree Shapley approaches with those based on Duke's approach [5]. We had to fall back to comparing how the different methods ranked features rather than the explanations themselves because there is no ground truth when it comes to explainability. It was difficult to argue for one explanatory approach over another; however, Duke's rankings seemed to agree more with the rankings based on Shapley values calculated under independence (which we saw was inaccurate in simulation studies), than with our proposed ctree based Shapley value estimation method.

References

1. Aas, K., Jullum, M., Løland, A.: Explaining individual predictions when features are dependent: More accurate approximations to shapley values. arXiv preprint arXiv:1903.10464 (2019)

2. Adadi, A., Berrada, M.: Peeking inside the black-box: A survey on explainable artificial intelligence (xai). IEEE Access **6**, 52138–52160 (2018)
3. Arellano-Valle, R.B., Branco, M.D., Genton, M.G.: A unified view on skewed distributions arising from selections. Canadian Journal of Statistics **34**(4), 581–601 (2006)
4. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and regression trees. Chapman and Hall (1984)
5. Chen, C., Lin, K., Rudin, C., Shaposhnik, Y., Wang, S., Wang, T.: An interpretable model with globally consistent explanations for credit risk. arXiv preprint arXiv:1811.12615 (2018)
6. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 785–794. ACM (2016)
7. Cramir, H.: Mathematical methods of statistics. Princeton U. Press, Princeton p. 500 (1946)
8. Elith, J., Leathwick, J.R., Hastie, T.: A working guide to boosted regression trees. Journal of Animal Ecology **77**(4), 802–813 (2008)
9. FICO: Explainable machine learning challenge (2018), https://community.fico.com/s/explainable-machine-learning-challenge
10. Fisher, R.A.: Statistical methods for research workers. In: Breakthroughs in statistics, pp. 66–70. Springer (1992)
11. Genz, A., Bretz, F.: Computation of Multivariate Normal and t Probabilities. Lecture Notes in Statistics, Springer-Verlag, Heidelberg (2009)
12. Hothorn, T., Hornik, K., Zeileis, A.: Unbiased recursive partitioning: A conditional inference framework. Journal of Computational and Graphical statistics **15**(3), 651–674 (2006)
13. Hothorn, T., Zeileis, A.: partykit: A modular toolkit for recursive partytioning in R. Journal of Machine Learning Research **16**, 3905–3909 (2015)
14. Lundberg, S.M., Erion, G.G., Lee, S.I.: Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888 (2018)
15. Lundberg, S.M., Lee, S.I.: A Unified Approach to Interpreting Model Predictions. In: Advances in Neural Information Processing Systems. pp. 4768–4777. Curram Associates Inc. (2017)
16. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc. (1993)
17. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2019)
18. Sellereite, N., Jullum, M.: shapr: An r-package for explaining machine learning models with dependence-aware shapley values. Journal of Open Source Software **5**(46), 2027 (2020)
19. Shapley, L.S.: A Value for N-Person Games. Contributions to the Theory of Games **2**, 307–317 (1953)
20. Štrumbelj, E., Kononenko, I.: An Efficient Explanation of Individual Classifications using Game Theory. Journal of Machine Learning Research **11**, 1–18 (2010)
21. Štrumbelj, E., Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. Knowledge and Information Systems **41**, 647–665 (2014)