Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Erik Holst Aasland

Shapley Values for dependent features using Divisive Clustering

Master's thesis in Industrial Mathematics
Supervisor: Kjersti Aas
June 2022

NTNU
Norwegian University of
Science and Technology

Erik Holst Aasland

Shapley Values for dependent features using Divisive Clustering

Master's thesis in Industrial Mathematics
Supervisor: Kjersti Aas
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Preface

This thesis is the work of the subject TMA4900 - Industrial Mathematics, Master's Thesis at the study programme Applied Physics and Mathematics at the Norwegian University of Science and Technology (NTNU) in Trondheim. The topic for the thesis was suggested by my supervisor Kjersti Aas.

I would like to thank Kjersti as a great source of information for the entire project. I would also like to thank her for her very helpful guidance in the finalisation of this project. Working with you has been a great pleasure.

I would also like to thank my friends who have made the hard work endurable. The moments with you gave me the energy and joy I needed during this long process.

Lastly I would like to thank my family, for their everlasting support and words of wisdom. Thank you for motivating me to always do my best. Without you I would not be where I am today.

Abstract

The aim of this thesis is to apply the new DivisiveShapApprox method to approximate SHAP values and to compare its performance to the KernelSHAP method.

There are now many methods to add interpretability to a machine learning model, and one of the most popular methods is the Shapley values. The Shapley values provide insight into the predictions made by a machine learning model by assigning each of the features used in the prediction a value of its contribution to the prediction. The SHAP values are the Shapley values for a single individuals' prediction. As the computation of Shapley values is very time consuming, one needs to use approximation methods in order to use Shapley values in practice.

In this thesis we compare two approximation methods for Shapley values. The first method we use is the KernelSHAP method, which is currently the most used approximation method. The second approximation method we use is the DivisiveShapApprox method, and a modified version of this method, DivisiveShapApproxNew. The approximation methods were compared over three sets of experiments.

The first set of experiments used data from a multivariate Gaussian distribution and a linear relationship between the output and the data. The second set of experiments also used multivariate Gaussian data, but a non-linear relationship between the output and the data, modelled with a random forest model. The final experiment used synthetic data, based on a real data set from the Norwegian financial service group DNB. On this data set a random forest model was fitted to predict the probability of default, and the SHAP values were computed using the different approximation methods.

The results of the experiments revealed that DivisiveShapApprox is too unstable for use, and that this is improved on in DivisiveShapApproxNew. The performance of DivisiveShapApproxNew is dependent on the relationship between the data and the output, and is only competitive when this relationship can be decomposed. For a simple relationship such as the linear model, the DivisiveShapApproxNew will outperform the KernelSHAP method, if both methods are given an equal short run time. As the run times of both methods increases, the error from the KernelSHAP method reduces faster than from the DivisiveShapApproxNew and the KernelSHAP is best if given enough computation time. In conclusion, the KernelSHAP is overall the better option.

Sammendrag

Formålet med denne oppgave er å anvende den nye metoden DivisiveShapApprox på å approksimere SHAP-verdier og å sammenlikne dens prestasjoner med KernelSHAP-metoden.

Det er for tiden mange metoder som gjør det mulig å tolke maskinlæringsmodeller, og én av de mest populære metodene er Shapley-verdier. Shapley-verdiene gir innsikt i prediksjonene til en maskinlæringsmodell ved å gi hver av variablene som er brukt for prediksjonen en verdi som beskriver variabelens bidrag til prediksjonen. SHAP-verdiene er Shapley-verdier for ett enkeltindivids prediksjon. Ettersom beregningen av Shapley-verdier er svært tidkrevende trenger man approksimasjonsmetoder for å kunne bruke Shapley-verdier i praksis.

I denne oppgaven sammenlikner vi to approksimasjonsmetoder for Shapley-verdiene. Den første metoden vi benytter er KernelSHAP-metoden, som for øyeblikket er den mest brukte approksimasjonsmetoden. Den andre approksimasjonsmetoden vi benytter er DivisiveShapApprox-metoden, samt en modifisert versjon av denne, DivisiveShapApproxNew. Approksimasjonsmetodene ble sammenliknet over tre sett med eksperimenter.

Det første settet med eksperimenter brukte data fra en multivariat Gaussisk fordeling og en lineær sammenheng mellom utfallet og dataene. Det andre settet med eksperimenter brukte også data med multivariat Gaussisk fordeling, men med en ikke-lineær sammenheng mellom utfallet og dataene, modellert av en random forest-modell. Det siste eksperimentet brukte syntetisk data, som er laget fra ekte data, som ble gitt av DNB. På dette datasettet blir en random forest-modell benyttet for å predikere sannsynligheten for mislighold, og SHAP-verdiene blir beregnet av de ulike approksimasjonsmetodene.

Resultatene fra eksperimentene avslører at DivisiveShapApprox er for ustabil til å være til bruk, men at dette er forbedret hos DivisiveShapApproxNew. Prestasjonene til DivisiveShapApproxNew avhenger av sammenhengen mellom utfallet og dataene, og metoden er kun konkurransedyktig når denne sammenhengen kan deles opp i mindre biter. For enkle sammenhenger, som en lineær modell, vil DivisiveShapApproxNew prestere bedre enn KernelSHAP, gitt at begge metodene har den samme korte kjøretiden. Når man øker kjøretiden til metodene vil feilen til KernelSHAP reduseres raskere enn hos DivisiveShapApproxNew og KernelSHAP vil være best, gitt lang nok kjøretid. Konklusjonen blir at KernelSHAP alt i alt er det beste valget.

Contents

1 Introduction **1**

2 Shapley values **4**

2.1 Shapley values in cooperative games 4

2.2 Shapley values for machine learning 5

2.3 Computational complexity of the Shapley values 5

2.4 SHAP 6

2.5 KernelSHAP 7

3 Cluster-decomposition approach **10**

3.1 A modification to the DivisiveShapApprox 12

4 Machine learning methods **14**

4.1 Linear model 14

4.2 Random forest 14

5 Experiments **17**

5.1 Linear model experiments 17

5.1.1 Experiment 1 19

5.1.2 Experiments 2 and 3 22

5.2 Random forest experiments 28

5.3 Synthetic data set from real data 31

6 Summary and further work **39**

References **40**

1 Introduction

This thesis is a further work of the author's Specialization Project (Aasland, 2022) on the divisive clustering approximation method DivisiveShapApprox (Corder and Decker, 2019). Parts of Sections 1, 2 and 3 are based on the previous work in this project.

The use of artificial intelligence (AI) and more specifically machine learning is experiencing a rapid incline in our society (Doshi-Velez and Kim, 2017). From being mostly hidden for the vast majority of people, the rise of AI has certainly caught the eye of the public as autonomous cars are starting to roam the roads, and reinforcement training algorithms such as AlphaZero are able to dominate against the winner of several computer chess championships, Stockfish 8, after training for only four hours (Silver et al., 2018). AI and machine learning are beginning to learn how to perform complicated tasks, and for the simpler and more defined tasks, humans are downright outperformed by these newcomers. Machine learning seems to be the way forward in numerous areas, and the amount of research going into the application of machine learning is substantial (a search on "using machine learning" after the year 2017 on Google Scholar returns over 1 million results).

Although machine learning is a great tool, which can help humans in both efficiency and in better understanding of certain tasks, the actual workings of the machine is often considered to be a black box. This can raise questions regarding the fairness of the machine learning model, if the model is able to perform its tasks correctly, and whether its decisions are based on criteria that humans would agree upon or not. E.g. we do not want an automatic loan acceptance based on machine learning that discriminates and is biased towards something deemed unfair by humans. Therefore it is important that machine learning being used for important tasks is either transparent enough that they can be controlled or so well-studied that the models are universally relied upon even though their inner workings may not be perfect (Doshi-Velez and Kim, 2017). In turn, this has created a need for ways to make the inner workings of machine learning models more transparent and making it possible to interpret what the model relies on when making its decisions.

The need for interpretable AI has led to several methods which provide different insights to the models decisions. The methods are usually divided into two groups, model-specific and model-agnostic. The model-specific methods are designed for specific classes of machine learning models, e.g. tree based models, and can only be used when the model that we are interested in explaining belong in this class. The model-agnostic methods do not assume any particular class of model, and only need the output of the model (often along with a data set for which the model makes its predictions). Thus model-agnostic models are more flexible than the model-specific methods, allowing the use of the same explanation method when switching between different machine learning models. Naturally, in order to not be redundant, the model-specific methods are faster and more precise than their model-agnostic counterparts. Another distinction is between global and local explanations. Global explanations relate the effects of the features to the entire distribution of the relevant data, e.g. the average effect a single feature has on the prediction of a data set. Local explanations are instead interested in explaining individual predictions. Before diving deeper into the theory of Shapley values and the approximation method presented by Corder and Decker (2019), some of the most well known model-agnostic methods in the field of explainable machine learning is presented briefly.

A relatively old method is the partial dependence plot (PDP). This plot shows how a single (or two) feature(s) affect the prediction of a machine learning model (Molnar, 2019). The PDP produce a line, which is the average prediction from the model for all values of the feature(s) of interest. The average prediction is made by marginalizing over all the remaining features taken as input in the model. In practice this means that for all the data points in the data set, only the feature of interest is changed, and then this new data point is fed to the model to get an output. The value on the PDP for a certain value x_s is then the average over all the new data points where this feature is set to x_s and the rest of the features are kept the same. This means that PDP assumes independence between the feature of interest and the other features taken as input in the model. The PDP is relatively easy to implement, and quite intuitive to interpret. It has some disadvantages with the aforementioned assumption of independence and the inability to be used on more than 2 features at once due to constraint on the visual representation of such a plot on the 2D surface of a computer screen or a paper. Another disadvantage of the PDP is that heterogeneous

effect can be hidden. Heterogeneous effects are effects that are not equal in the entire feature space, but differ between different parts of the feature space. If a part of the data set would have a positive effect on the output as the feature increases, and some other part would have a negative effect as the feature increases, then the PDP could appear with a horizontal line and the chosen feature would falsely seem to have no effect on the output.

A method which works in much the same way as the PDP is the individual conditional expectation (ICE) method. Instead of plotting the average output over all the instances for the particular value of a feature, a separate line for each of the instances' output is included. Being so similar to the PDP, the interpretation and level of difficulty on implementation is similar between the two methods. Some differences however, is that ICE is able to give some local information on the model, and the method can thus show heterogeneous effects that could be hidden in the PDP. Disadvantages of the ICE curves are that it can only be used on one feature at a time, that a too big number of instances leave a too overcrowded plot and that it might be difficult to see the average effect. Some of these problems have solutions like only including a set number of instances and adding a PD line on the plot as the ICE curves. ICE also assume independence between the feature of interest and the other features.

Both PDP and ICE assume independence, and quickly run into unrealistic data if features are in fact correlated. A method which achieve the same type of interpretation as the PDP, but handles dependent features, is the accumulated local effects (ALE). Part of the way ALE handles dependent features is by creating intervals for the feature of choice, and dividing the data points of the data set into which of these intervals they belong, such that the data points used to get outputs from the model are far more likely to be realistic. Advantages of ALE include being unbiased when features are correlated, and also being a faster method than the PDP. Molnar (2019) lists quite a few disadvantages of the ALE, some of which are that one can not interpret the effect of the feature across intervals and that the ALE is not able to work together with an ICE plot. Despite ALE being far more complex, thus more difficult to implement, Molnar recommends the method over the PDP in most situations.

The three mentioned methods, PDP, ICE and ALE, all generate functions of only one (or two) features which display some interpretation of the effect this (these) features have on the output of the model. Other methods use surrogate models to get an interpretation on the effect of more or all the features. Surrogate models are models that are more interpretable than the actual model that we are looking into, and which approximate the actual model as closely as possible, while often times remaining simple. The global surrogate method creates a surrogate model (e.g a linear model or a decision tree) for the full distribution of the data. The local interpretable model-agnostic explanations (LIME) method is a local surrogate method. Here the surrogate model is fitted around a single data point, which is done by weighting the data points according to their proximity to the data point which is desired to explain. The advantage of the surrogate methods includes that if they are simple enough (e.g. lasso or short trees), then the resulting models are very fast to interpret. Some disadvantages on the other hand are that the surrogate models must be close enough to the actual model in order to give the right picture, and there is no clear threshold for when a surrogate model is good enough (Molnar, 2019). For the LIME method, the choice of which proximity measure to use may also greatly impact the results.

All the above methods describe how the outcome of the model changes given certain values for the features. Other methods are less interested in mapping the model to a function, and care more about how important each feature is for the output. One method which aims to describe the global importance of each feature is the permuted feature importance method. In this method, the importance of a feature is defined as the error of the predicted values from the model when the values of this feature is permuted between the data points of the data set. Permuted feature importance yields results that are quite intuitive and tidy. One of the disadvantages of the method is that in order to measure the error, the actual values the model is trying to predict needs to be available, which might not be the case if you are just given an already built model.

Shapley values is another method which assigns each feature a value (Shapley, 1953) and is the method in focus in this project. Instead of importance, the Shapley value should be interpreted as the feature's contribution to the prediction. Shapley values can be used for the local contribution,

then called SHAP values. SHAP values explain the contribution of each feature on the outcome of the model for a specific data instance. Shapley values can also be used for global contribution by assigning each feature their contribution to the models total R^2, which is called Shapley regression. A big difference to the other mentioned interpretation methods is that the Shapley value is the only method based on solid theory (Molnar, 2019). However, the computation of the exact Shapley values for a model is extremely computationally complex, and in general one has to rely on approximations methods for the Shapley values in order to use Shapley values in practice.

A final model-agnostic method to mention is the counterfactual explanations. Counterfactual explanations have a very different aim than the other methods discussed in this section which are all concerned with explaining why a certain prediction was made. The counterfactual explanations do not care why a prediction was made, only on how the prediction can be changed. It is based on the human-friendly question: "What is the smallest change needed to make the prediction reach a certain level?". This sort of question is likely to be the most important for people who encounter an automatic (machine learning trained) approval system and experience to be declined. Using counterfactual explanations as the explanation method could then be the most reasonable choice as the intuition of its results is very clear. The problem with using counterfactual explanations is determining which "small change" in the features is considered as the smallest change for the user, and thus how one should measure distance between cases, and also which solutions to present when multiple solutions are found (Molnar, 2019).

The aim of this thesis is to compare the performance of the approximation method DivisiveShapApprox by Corder and Decker (2019) to the KernelSHAP method (Lundberg and Lee, 2017), with respect to approximating SHAP values with dependent features. The thesis is structured as follows:

Section 2 introduces the theory of the Shapley value, from its origin in cooperative game theory to its application for machine learning models. Here we also present the SHAP values and the approximation method KernelSHAP. Section 3 describes the DivisiveShapApprox method to approximate Shapley values, as well as a new modified version of the method. In Section 4 we introduce the machine learning methods used in our experiments, namely the linear model and the random forest model. Section 5 describes and discusses the experiments for which the approximation methods are tested on. The experiments include some simulated experiments on linear models and a random forest model, in addition to a synthetic data set from the Norwegian financial service group DNB. In Section 6 we draw a conclusion from the results of the experiments in Section 5 and discuss some future work.

2 Shapley values

The Shapley value is originally a concept from cooperative game theory (Shapley, 1953). By reformulating the resulting model from some machine learning algorithm, it is possible to apply the concept of Shapley value to the world of machine learning, and by doing so obtaining some information regarding the contribution of each parameter in the model. In this section, the definition of the Shapley value from cooperative game theory is presented first. Then, some of the ways to apply it to machine learning are described. Lastly, the computational complexity of finding Shapley values is discussed and some approximation methods to improve the computation time are presented.

2.1 Shapley values in cooperative games

A cooperative game consist of a set of players $N = \{1, \cdots, n\}$ and the characteristic function $v : 2^N \to \mathbb{R}$. The characteristic function $v(S)$ can then represent the payoff a coalition $S \subseteq N$ get from cooperating. The coalition with all the players N, is often called the grand coalition. The players in the cooperative game should be considered as self-interested and are only cooperating as to maximise their own profit. The problem that arise is then how the total payoff should be distributed among the players.

Shapley value provides one way of distributing the total payoff between players. The Shapley value is the unique solution to the aforementioned problem, where also four desirable fairness properties are fulfilled. These properties are (Shapley, 1953):

- **Efficiency:** The resulting additional payoff from the cooperation should be distributed fully among the players. In mathematical notation this can be written as

$$v(S) = \phi_0 + \sum_{i \in S} \phi_i(v).$$

  where $\phi_0 = v(\emptyset)$ can be interpreted as the base payoff, which is achieved even when no players are playing. The players thus only share the difference $v(S) - v(\emptyset)$ which they actually contribute to.

- **Symmetry:** Two players who contribute equally to any coalition should be given the same amount.

$$\begin{aligned} v(S \cup \{i\}) = v(S \cup \{j\}) \qquad \forall \quad S : \{i, j\} \notin S \\ \implies \phi_i(v) = \phi_j(v) \end{aligned}$$

- **Null player:** A player who do not contribute in any coalition should not receive anything.

$$v(S \cup \{i\}) = v(S) \forall S : \{i\} \notin S \implies \phi_i(v) = 0$$

- **Linearity:** If the characteristic function can be expressed as the sum of two other characteristic functions, then the relationship should be linear. This means that

$$\phi_i(av + bw) = a\phi_i(v) + b\phi_i(w)$$

  where v and w are characteristic functions.

The formula for the Shapley value in a cooperative game is then given as

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \big(v(S \cup \{i\}) - v(S)\big), \quad i = 1, \cdots, n. \tag{1}$$

The Shapley value can be interpreted as the marginal contribution of player i to the payoff of a coalition (Colini-Baldeschi et al., 2018). In Equation (1) we can note how $\frac{|S|!(n-|S|-|)!}{n!} =$

$\frac{1}{n}\frac{|S|!(n-1-|S|)!}{(n-1)!} = \frac{1}{n}\binom{n-1}{|S|}^{-1}$. This means that for each number of players $|S|$, there are $\binom{n-1}{|S|}$ combinations where the contribution of i is summed, and then the average is taken by dividing this sum by the number of combinations. Finally the average over each of these n averages with $|S|$ from 0 to $n-1$ is computed to get the Shapley value.

2.2 Shapley values for machine learning

In the machine learning setting, the players are the variables of the machine learning model in question (Lundberg and Lee, 2017). For the characteristic function there are infinitely many choices, as one can choose any function $v : 2^N \to \mathcal{R}$ assigning a value to any subset of the input parameters of the model. Depending on the choice of characteristic function the corresponding Shapley value have different interpretations.

In the case of linear models, the characteristic function $v(S) = R^2$ leads to what is called Shapley regression (Lundberg and Lee, 2017). $v(S)$ is then the corresponding R^2 of the fitted linear model using only the covariates included in S. The results from the Shapley regression can then be interpreted as the relative importance of each variable in the model. It should be noted that these values describe the variables' relative global importance, that is, their importance for the entire data set.

Shapley values can also be used to determine the local importance of covariates on predictions. One way to achieve this, as seen in Aas et al. (2019), is with the characteristic function $v(S) = \mathrm{E}[f(\boldsymbol{x})|\boldsymbol{x}_S = \boldsymbol{x}_S^*]$, where $f(\boldsymbol{x})$ is the prediction from the model in question given covariate values $\boldsymbol{x}$, and $\boldsymbol{x}_S$ is a subset of the covariates. The Shapley value is then found for a given individual with covariates values $\boldsymbol{x}^*$. The Shapley values for this characteristic function are often referred to as SHAP (SHapley Additive exPlanations) values and it is this characteristic function which will be used in this thesis.

2.3 Computational complexity of the Shapley values

The main drawback for the practical use of the Shapley value in machine learning is the computational complexity. Computing the Shapley value using the definition in Equation (1) includes calling the characteristic function for every subset of players at least once. With optimal design, this means that the run-time for computing the Shapley value is $O(2^n)$, as there are 2^n possible combinations of n players or covariates. Computationally heavy characteristic functions will add further run-time complexity, and is another possible drawback of the Shapley value method. As a result of the general formulation for the Shapley value having such a high computational complexity, there is a need for faster methods. This has given rise to approximation methods such as KernelSHAP, which will be further explained in Section 2.5, and Shapley Sampling Values (Štrumbelj and Kononenko, 2010), which approximates the Shapley value by a Monte Carlo approach. The divisive clustering method proposed by Corder and Decker, DivisiveShapApprox approximates Shapley values of games with recognizable equivalence, and is explained in detail in Section 3. In Aasland (2022) the DivisiveShapApprox was found to be incredibly fast even for numerous players, but lacking in precision when there was dependence between many of the players.

Exact formulas for the Shapley value, with less complexity, has been discovered for some specific game representations as well, providing the best option when such games arise. In the specialization project leading up to this thesis (Aasland, 2022) it was shown that the features game (Corder and Decker, 2019; Soufiani et al., 2014) and Variance game (Colini-Baldeschi et al., 2018) are two game representations which have exact and explicit formulas for Shapley values. In this thesis, the focus is on SHAP values for predictive models. Simple and exact formulas for the Shapley values are also known in certain cases in this case. In what follows we will describe the cases.

For the already easy to interpret linear regression model, $f(\boldsymbol{x}) = \boldsymbol{x}^\top\boldsymbol{\beta}$, along with the characteristic function for local importance $v(S) = \mathrm{E}[f(\boldsymbol{x})|\boldsymbol{x}_S = \boldsymbol{x}_S^*]$, the Shapley value can be calculated easily through $\phi_j = \beta_j(x_j^* - \mathrm{E}[x_j])$, if the covariates are independent (Aas et al., 2019).

When using a linear model with dependent features there is no such general exact formula for the Shapley values. However the computation of the characteristic function can be simplified in this case by (Aas et al., 2019)

$$\begin{aligned}
v(S) &= \int f(\boldsymbol{x}_{\bar{S}}, \boldsymbol{x}_S^*) p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x}_{\bar{S}} \\
&= \int \left(\sum_{j \in \bar{S}} \beta_j x_j + \sum_{j \in S} \beta_j x_j^* \right) p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x}_{\bar{S}} \\
&= \sum_{j \in \bar{S}} \beta_j \int x_j p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x}_{\bar{S}} + \sum_{j \in S} \beta_j x_j^* \int p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x}_{\bar{S}} \\
&= \sum_{j \in \bar{S}} \beta_j \mathrm{E}[x_j | \boldsymbol{x}_S = \boldsymbol{x}_S^*] + \sum_{j \in S} \beta_j x_j^* \\
v(S) &= f(\mathrm{E}[\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*], \boldsymbol{x}_S^*). \qquad (1)
\end{aligned}$$

This is useful as long as $\mathrm{E}[\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*]$ is feasible to obtain, which is generally not the case.

2.4 SHAP

The characteristic function for the SHAP values is as previously mentioned

$$v(S) = \mathrm{E}[f(\boldsymbol{x}) | \boldsymbol{x}_S = \boldsymbol{x}_S^*], \qquad (2)$$

where $f(\boldsymbol{x})$ is the prediction of the model in question for an individual with covariate values $\boldsymbol{x}$, and $\boldsymbol{x}^*$ is the covariate values of the individual whose prediction we want to explain. Following the notation by Aas et al. (2019), the complement set to S is denoted $\bar{S}$ such that $N = \{S \cup \bar{S}\}$. From the definition of expected values we get

$$\begin{aligned}
\mathrm{E}[f(\boldsymbol{x}) | \boldsymbol{x}_S = \boldsymbol{x}_S^*] &= \int f(\boldsymbol{x}) p(\boldsymbol{x} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x} \\
&= \int f(\boldsymbol{x}_{\bar{S}}, \boldsymbol{x}_S^*) p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*) d\boldsymbol{x}_{\bar{S}}, \qquad (3)
\end{aligned}$$

where $p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*)$ is the conditional distribution of $\boldsymbol{x}_{\bar{S}}$ given the covariates in S of the individual of interest.

The original approach by Lundberg and Lee (2017) is to assume independence between $\boldsymbol{x}_S$ and $\boldsymbol{x}_{\bar{S}}$, which reduce the conditional distribution $p(\boldsymbol{x}_{\bar{S}} | \boldsymbol{x}_S = \boldsymbol{x}_S^*)$ to the marginal distribution $p(\boldsymbol{x}_{\bar{S}})$. This way, the expected value can be estimated by the Monte Carlo method, simulating the marginal distribution of $X_{\bar{S}}$ by drawing directly from the observed data.

When dealing with real data, such an assumption of independence is often not justified, and some covariates can be quite heavily correlated. Thus, the conditional distribution can not be approximated by the marginal distribution, and the aforementioned method is not possible. Possible approaches to handle dependent covariates in the characteristic function are discussed by Aas et al. (2019) and Redelmeier et al. (2020).

A relatively simple approach to incorporate conditional probabilities is by assuming that the covariates come from a multivariate normal distribution, $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The parameters for such a distribution can be estimated from the data set. For a coalition S and its complement $\bar{S}$, the parameters of the multivariate normal distribution can be rewritten as

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{\bar{S}} \\ \boldsymbol{\mu}_{S} \end{bmatrix}$$
$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\bar{S}\bar{S}} & \boldsymbol{\Sigma}_{\bar{S}S} \\ \boldsymbol{\Sigma}_{S\bar{S}} & \boldsymbol{\Sigma}_{SS} \end{bmatrix}.$$

The conditional distribution of the covariates not in S will then also be multivariate normal (Härdle and Simar, 2019), i.e. $\boldsymbol{X}_{\bar{S}}|\boldsymbol{X}_S = \boldsymbol{x}_S \sim \mathcal{N}(\boldsymbol{\mu}_{\bar{S}|S}, \boldsymbol{\Sigma}_{\bar{S}|S})$, where

$$\boldsymbol{\mu}_{\bar{S}|S} = \boldsymbol{\mu}_{\bar{S}} + \boldsymbol{\Sigma}_{\bar{S}S}\boldsymbol{\Sigma}_{SS}^{-1}(\boldsymbol{x}_S - \boldsymbol{\mu}_S) \tag{4}$$

$$\boldsymbol{\Sigma}_{\bar{S}|S} = \boldsymbol{\Sigma}_{\bar{S}\bar{S}} - \boldsymbol{\Sigma}_{\bar{S}S}\boldsymbol{\Sigma}_{SS}^{-1}\boldsymbol{\Sigma}_{S\bar{S}}. \tag{5}$$

To evaluate the characteristic function defined in Equation (2) analytically using the integral defined in Equation (3), both terms of the integrand would have to be analytical. The prediction of the model, f, will in many cases not be an analytical function, or at least convoluted enough to make exact integration futile. Assuming that the data is multivariate Gaussian, the integral in Equation (3) can be approximated using Monte Carlo integration, by sampling from the now known conditional distribution with parameters from Equation (4) and Equation (5). This means that

$$v(S) \approx \frac{1}{K}\sum_{k=1}^{K} f(\boldsymbol{x}_{\bar{S}}^k, \boldsymbol{x}_S^*), \tag{6}$$

where $\boldsymbol{x}_{\bar{S}}^k$ is one of K random samples from the multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_{\bar{S}|S}, \boldsymbol{\Sigma}_{\bar{S}|S})$. Note that computing the characteristic function as in Equation (6) is significantly more time consuming than when having a simple formula for v, especially considering K needs to be large enough to get a somewhat consistent evaluation during re-runs.

2.5 KernelSHAP

One of the most well-known approximation methods of the Shapley value is the KernelSHAP method (Lundberg and Lee, 2017). KernelSHAP was primarily an approximation method for SHAP values, hence its name, but the concept has no limitations to specific characteristic functions. The KernelSHAP method utilizes a different formula for the Shapley values. This formula is the result of the following weighted least squares formulation of the Shapley value problem:

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi}} \sum_{S \subseteq N} \left(v(S) - \left(\phi_0 + \sum_{j \in S} \phi_j \right) \right)^2 \cdot k(n, S), \tag{7}$$

where $\boldsymbol{\phi}$ is the set of all the Shapley values, and the kernel $k(n, S)$ is defined as

$$k(n, S) = \frac{n-1}{\binom{n}{|S|}|S|(n-|S|)}. \tag{8}$$

Solving the weighted least squares problem in Equation (7) is proven to lead to the same Shapley values as using Equation (1). From Equation (8) we have that $k(n, \emptyset) = k(n, N) = \infty$, which in practice means that Equation (7) contains the two constraints $\phi_0 = v(\emptyset)$ and $\phi_0 + \sum_{j \in N} \phi_j = v(N)$. For any practical use, using e.g. $k(n, \emptyset) = k(n, N) = 10^6$ is sufficient for the results to be very good approximations.

Now, this optimisation is still $O(2^n)$ and can barely be considered an approximation, but the trick of the KernelSHAP is to approximate this optimisation problem. Since the kernel give a very small weight for most of the subsets S, each of these subsets impact the overall minimisation very little as n grows larger. The way KernelSHAP uses this property is by sampling only d number subsets D, $d = |D|$, with replacement, from the set of all possible combinations, with each subset having a probability of being picked following the kernel weights in Equation (7). As the probability of a subset being sampled now follows the kernel weights, the sampled subsets will be weighted equally. The two subsets with infinite kernel values, $\emptyset$ and N, are not included in the sampling process, but can be dealt with in one of two ways. They can both be included once in the sampling and be given weights of e.g. 10^6 or be set as constraints. If the weights are set to 10^6, the new minimisation problem becomes

$$\boldsymbol{\phi} = \arg\min_{\boldsymbol{\phi}} \sum_{S \in D} \left(v(S) - \left(\phi_0 + \sum_{j \in S} \phi_j \right) \right)^2 \cdot w(S), \tag{9}$$

which can be regarded as a weighted linear regression, with well known solutions. $w(S)$ is the weight of subset S and is 10^6 for $\emptyset$ and N and 1 otherwise. This reduces the complexity of the computation from $O(2^n)$ to $O(d)$, as the number of subsets is reduced from 2^n to d. If the $\emptyset$ and N subsets are given as constraints, these need not be included in the sampling, and Equation (9) is a simple (non-weighted) linear regression.

In contrast to the previous work in the specialization project (Aasland, 2022), the two constraints, $\phi_0 = v(\emptyset)$ and $\sum_{j=0}^{n} \phi_j = v(N)$ are in this thesis incorporated exactly. Firstly, as the SHAP values explain the deviation of the individual prediction from the average prediction of the data set, the characteristic function can instead report this difference, $v^*(S) = \mathrm{E}[f(\boldsymbol{x})|\boldsymbol{x}_S = \boldsymbol{x}_S^*] - \mathrm{E}[f(\boldsymbol{x})]$. From the linearity property of the Shapley value, this can be seen as the linear combination of two characteristic functions, and the resulting Shapley value will thus also be the linear combination of the corresponding Shapley values. The Shapley values for the second term, $\mathrm{E}[f(\boldsymbol{x})]$ is $\phi_0 = v(\emptyset)$ and $\phi_i = 0$ for all other i, because $\mathrm{E}[f(\boldsymbol{x})]$ is a constant. This means that $v^*(S)$ will yield the same Shapley values as v, except for ϕ_0 which will always be exactly 0, and the first constraint is satisfied.

The second constraint, $\sum_{j=0}^{n} \phi_j = v(N)$, can be satisfied by explicitly setting $\phi_n = v(N) - \sum_{i=1}^{n-1} \phi_i$. Each occurrence of ϕ_n is thus replaced by $v(N) - \sum_{i=1}^{n-1} \phi_i$ in Equation (9). Note that the two data points, $\emptyset$ and N, are no longer needed in D, as these differences are always just 0 with our new constraints. The reported ϕ_n is consequently computed from the other Shapley values from the linear regression.

The kernel from Equation (8) for a specific coalition S in itself does not show the probability of this coalition being sampled directly. This is due to the fact that the kernel is not scaled, and the proper probability of a coalition S being sampled can be written as

$$P(\text{Sampling } S) = \frac{C}{\binom{n}{|S|}|S|(n - |S|)}, \tag{10}$$

for some C such that the sum of all the probabilities add to 1. For a relatively large number of covariates, computing the probabilities for all the 2^n possibilities defeats some of the purpose of the KernelSHAP method. One way of avoiding this complication is by noticing that the probabilities for all sets S with an equal number of covariates is the same. The number of coalitions with k members from the total of n covariates is $\binom{n}{k}$, and the probability of sampling a coalition with k is thus $\frac{C}{k(n-k)}$. The resampling can therefore be done by first drawing the number of covariates, i.e. 6, then adding 6 random (uniformly) covariates to the coalition. This reduces the number of probabilities required from 2^n to $n - 1$.

The number of resampled subsets in D, d, affects the run-time, but also the precision of the approximation. A general agreed upon expression for d was not found in the research for this

thesis. Covert and Lee (Covert and Lee, 2020) cast some light on the subject, finding empirical evidence that the variance of KernelSHAP is $O\left(\frac{1}{d}\right)$, but still offer no suggestion as to the best choice of d.

The KernelSHAP method was originally used for approximating SHAP values, which is also the intention in this thesis, but the approach will not be the same as in Lundberg and Lee (2017). As mentioned in Section 2.4, the original approach of KernelSHAP to estimate Equation (3) is to assume independence between $\boldsymbol{x}_S$ and $\boldsymbol{x}_{\bar{S}}$, and thus simulate the marginal distribution of $X_{\bar{S}}$ by drawing directly from the observed data. As the experiments in Section 5 use data with high correlation between features, this approach is mostly avoided. Instead, the KernelSHAP will primarily follow the approach of Aas et al. (2019) when computing the characteristic function in Equation (2), by assuming data from a multivariate Gaussian distribution. Further use of the term KernelSHAP thus reference the computation of SHAP values through Equation (9), and not how the characteristic function is computed.

The popularity of the KernelSHAP approximation is in large due to the fact that it is a model-agnostic approximation method. Another advantage of the KernelSHAP method compared to using Equation (1) is that it computes the approximated (or exact, if one use Equation (7)) Shapley value through the matrix multiplication $H\boldsymbol{v}$, where H is the hat matrix from the (weighted) linear regression and $\boldsymbol{v}$ is the vector containing the corresponding values of the characteristic function v for the subsets included, $\boldsymbol{v} = (v(S)_{S \in D})$. This means that for a new characteristic function v_{new}, the Shapley value can be computed quickly through the matrix multiplication $H\boldsymbol{v}_{\text{new}}$, and H does not need to be computed again, while with Equation (1) the entire computation would have to be done again.

3 Cluster-decomposition approach

The approximation method DivisiveShapApprox by Corder and Decker (2019) is the main focus of this project. Similar to KernelSHAP, DivisiveShapApprox is also model-agnostic, as it uses only the output of the model, and can thus be used for any machine learning model. DivisiveShapApprox differs from KernelSHAP however, as DivisiveShapApprox assumes to be used on a game where the features (players) have a set of attribute values and that the characteristic function of the game, v, is a function of these attributes. It also assumes that features with similar sets of attribute values will get similar values from the characteristic function v. `DivisiveShapApprox` exploits this assumption by grouping features based on similar attributes. This means that unlike KernelSHAP, DivisiveShapApprox can not be used in any scenario, despite being model-agnostic, because it needs a measure of similarity between all features as well the output of the model.

A cooperative game is said to have a multi-issue representation when it can be divided into T independent subgames (Corder and Decker, 2019), and the characteristic function of the game can be expressed as the sum of some individual characteristic functions v_j for the subgames, that is $v(S) = \sum_{j \in T} v_j(S)$. It follows from the linearity property of the Shapley value in 2.1 that the Shapley value for a multi-issue represented game is the sum of the Shapley value ϕ_{ji} for each of the independent subgames. This changes the complexity from $O(2^n)$ to $O(\sum_{j \in T} 2^{|C_j|})$, where $|C_j|$ is the number of players which impact the value of v_j. This means there is a reduction in complexity if the characteristic function has a multi-issue decomposition where every subgame has a characteristic function v_j which only depend on some of the players. Exact multi-issue decompositions are rarely possible, but the DivisiveShapApprox method applies the idea with an approximate multi-issue decomposition.

Another form of game representation are agent type games. In these games the players are referred to as agents, and there are different equivalence classes connected to the agents. Agents that contribute equally to every coalition are called strategically equivalent, and by the symmetry property of the Shapley value in 2.1 such agents have an equal Shapley value. In games where agents have attributes, agents with all equal attributes are called representationally equivalent agents. These equivalence classes have given rise to the agent type representation called recognizable equivalence, where the characteristic function is assumed to depend on the agents attributes.

The DivisiveShapApprox method by Corder and Decker (2019) assumes games with recognizable equivalence. By doing so, the method opens for a partitioning of the grand coalition N based only on the agents attributes. This partitioning will end with l leaf coalitions with maximum size $log_\beta(n)$, where β is a tuning parameter for the method, and can be done by an arbitrary clustering algorithm. For each of these leaf coalitions, containing no more than $log_\beta(n)$ agents, the Shapley value will be computed exactly for this subgame. As noted by Corder and Decker (2019), the characteristic functions used in these subgames should still be the same characteristic function as in the full game, which does not actually decompose v into individual v_i for the different coalitions, as in a proper multi-issue-decomposition, but instead could be described as an ϵ-approximate decomposition, $v = \epsilon + \sum_{C_k \in l} v(C_k)$.

When joining Shapley values from two branches of a partition, the Shapley value from each of the branches is scaled such that the sum of the Shapley value approximations a_ϕ for the agents in the current coalition is equal to the total payout for the subgame with only these agents. In other words, as the grand coalition is rejoined from its partitions, the Shapley value from the branches are scaled by the synergy rate γ, the rate in which the value change as the subcoalitions are joined, which means that $\gamma = v(S) / \sum_i v(S_i)$, with S_i being the subcoalitions from the partitioning of S. This scaling process is responsible for ensuring that the approximations made by DivisiveShapApprox satisfy the efficiency property of the Shapley value from Section 2.1. Corder and Decker (2019) also prove that the method satisfies the symmetry property of the Shapley value as well as the property of the null player when not all leaf coalitions have value $v(S_i) = 0$, but $v(N) \neq 0$, or as long as null players are more similar than any non-null player are to a null player.

The process of partitioning the grand coalition N into leaf coalitions, exact computation of Shapley values for the leaf coalitions and backwards feeding of Shapley values up to the grand coalition is

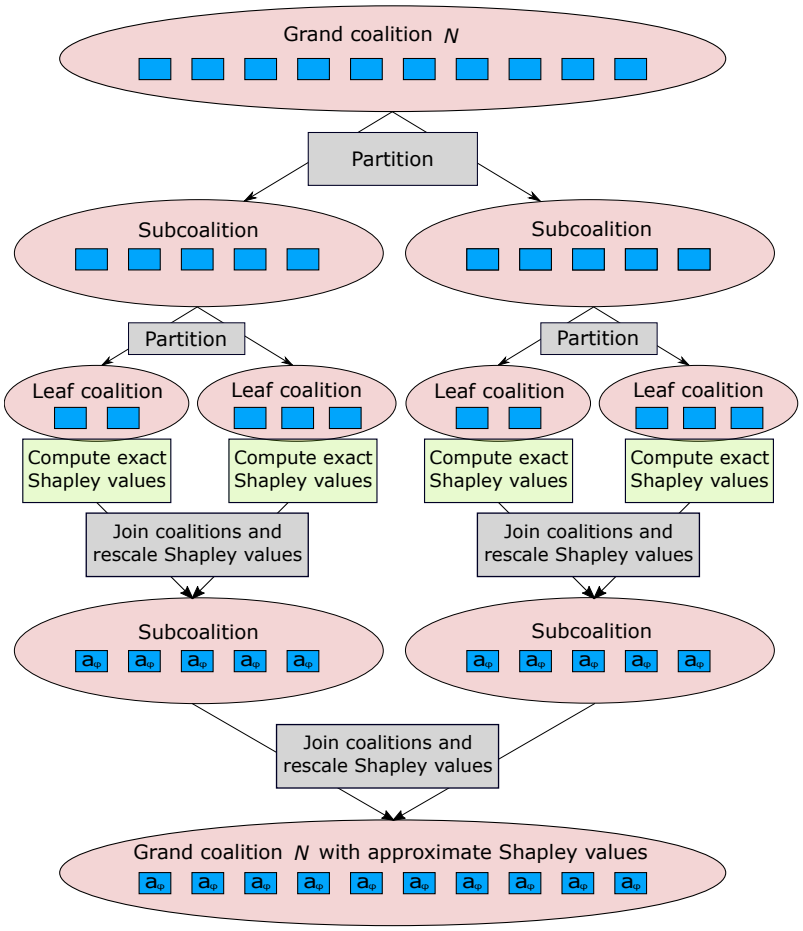


Figure 1: The concept of the DivisiveShapApprox method. The grand coalition with all players go through several partitions by clustering methods until there are only leaf coalitions with less than $\log_\beta(n)$ players in each. In the leaf coalitions, the exact Shapley values for these subgames are computed. The approximated Shapley values for each coalition, a_ϕ is subsequently computed from its own subcoalitions until the Grand coalition is reached.

shown in Figure 1. The pseudo code for DivisiveShapApprox is presented in detail by Corder and Decker (2019) p.5.

The run-time complexity of the method depends on the tuning parameter β. A high value for β means the leaf coalitions are smaller, which leads to the exact computation of the Shapley value in the leaf coalitions being faster, while a low β leads to fewer partitions and less error from the approximation. Thus β also control the trade-off between run-time and error. Corder and Decker (2019) prove the run-time complexity of DivisiveShapApprox when using k-means clustering with $k = 2$ to be $O(n^{\log_\beta(2)+1})$ and with best case run-time $\Omega\left(\frac{n^{\log_\beta(2)+1}}{\log_\beta(n)}\right)$. In their experiments, Corder and Decker (2019) use k-means clustering as partition method and $\beta = n^{1/\sqrt{n}}$. This construction of the β results in the maximum size of leaf-clusters $log_\beta(n) = \sqrt{n}$.

k-means clustering is a well known clustering algorithm, where the number of clusters k is decided up front. The algorithm starts by choosing k random data points from the data set to serve as initial cluster centers, or centroids. From here, two steps are repeated until convergence, which is when no changes are made in one iteration. First, each data point is assigned to the cluster with the closest centroid. Second, the clusters' new centroids are computed as the mean position of all the data points in that cluster.

In the experiments of this thesis it is found more appropriate to use another clustering approach called hierarchical clustering. Hierarchical clustering is a set of clustering methods where the clusters are created in a hierarchy. On the bottom all objects are their own cluster. As one rises

through the hierarchy, clusters are joined together all the way to the top where there is just one cluster. The results are typically represented visually using dendrograms. There are generally two classes of hierarchical clustering methods, agglomerative methods and divisive methods (Maimon and Rokach, 2005). Agglomerative methods construct the hierarchical clustering bottom-up, joining together clusters from the bottom until there is just one cluster, while divisive methods constructs the hierarchical clustering top-down, starting out with one cluster which is divided into two clusters, repeated until all clusters have only one instance.

The clusters in the hierarchical clustering are decided from some similarity measure between instances (Maimon and Rokach, 2005), and there are different ways to use this similarity measure when comparing two clusters. Single-link clustering considers the similarity between two clusters as the highest similarity between two instances from the different clusters, and is sometimes called nearest-neighbor. Complete-link clustering is another approach and it considers the similarity between two clusters to be the smallest similarity across the clusters, and is similarly called furthest-neighbor.

To summarise, KernelSHAP and DivisiveShapApprox achieve their approximations in completely different manners. KernelSHAP approximates the Shapley values by the assumption that most coalitions do not matter much for the Shapley values, and that having just a relatively small subset of coalitions and their respective payouts, $v(S)$, is enough to get good approximations of the Shapley values. The DivisiveShapApprox method assumes that the game itself is nearly a composition of smaller subgames. DivisiveShapApprox approximates the Shapley values by computing the Shapley values in these subgames, and by using the linearity property of the Shapley value, that the Shapley values for the full game can be represented by the Shapley values from the subgames, adjusted by certain scaling factors.

It should be noted that KernelSHAP and DivisiveShapApprox use the same characteristic function, namely the characteristic function of the full game. The fact that DivisiveShapApprox assumes v to be a function of the features' attributes is simply a condition for the method to have a proper foundation, while KernelSHAP has no such condition. In their paper Corder and Decker (2019) implicitly state that $v(\emptyset) = 0$. This is dealt with by instead using a modified characteristic function $v^*(S)$, mentioned in Section 2.5, which is simply a relocated version of the characteristic function $v(S)$. Note that adding a constant to the characteristic function $v(S)$ will only impact ϕ_0, and the other Shapley values will remain the same as before. Thus, the interpretation of the Shapley values for the features when using the relocated $v^*(S)$ is exactly the same as when using $v(S)$. For SHAP values, $v(\emptyset) = \mathrm{E}[f(\boldsymbol{x})]$, and in the setting of the later experiments $\mathrm{E}[f(\boldsymbol{x})]$ is the average prediction for the data set, when assuming the data set is representable for the entire population. This also means that one can find ϕ_0 for the original setting with characteristic function $v(S)$ as $\phi_0 = \overline{\boldsymbol{f}(\boldsymbol{X})}$, where $\boldsymbol{X}$ is the entire data set. From this point on, $v^*(S)$ will be used as characteristic function, and will be referred to as $v(S)$.

3.1 A modification to the DivisiveShapApprox

A problem which often occurs in the experiments in Section 5 is a huge error in the approximations made by DivisiveShapApprox for some individuals. This error is caused by the scaling step in the method, which in some cases is way off. Because the scaling factor, γ, is decided by the characteristic function on the subcoalitions, the approximations are highly vulnerable to mistakes here. Such mistakes are particularly likely to happen when subcoalitions are valued close to 0, i.e. $v(S) \approx 0$.

An example can illustrate this more clearly. Consider the small problem where the grand coalition N is only partitioned once into the two leaf coalitions S_1 and S_2. Since the decomposition of v is not perfect, we have that $v(N) = v(S_1) + v(S_2) + \epsilon$. The synergy rate when joining these two coalitions is then

$$\gamma = \frac{v(N)}{v(S_1) + v(S_2)} = \frac{v(S_1) + v(S_2) + \epsilon}{v(S_1) + v(S_2)} = 1 + \frac{\epsilon}{v(S_1) + v(S_2)}.$$

This is of course an intended feature, but the method becomes rather unstable whenever $|v(S_1) + v(S_2)| << \epsilon$. This is particularly likely to happen when the decomposition of v is not very good, making ϵ relatively big. In addition, since $v^*(S)$ is centered around 0, as described in Section 2.5, getting $v(S_1) + v(S_2) \approx 0$ is not unlikely. Say $v(S_1) + v(S_2) = 0.2\epsilon$, this gives a synergy rate of $\gamma = 1 + \frac{\epsilon}{0.2\epsilon} = 6$. Thus, although the exactly computed Shapley values in each leaf coalition might be close to the actual value for the full game, the scaling will completely ruin the approximation. The results are even worse if the error is big enough to flip the sign, e.g. $v(S_1) + v(S_2) = -0.5\epsilon$. This results in a synergy rate of $\gamma = 1 - \frac{\epsilon}{-0.5\epsilon} = -1$, which means that the Shapley values from the leaf-coalitions will be completely flipped in the approximation of the Shapley values for the full game. This would make the biggest contributor in a leaf coalition the biggest inhibitor in the full game.

The mistakes due to heavy scaling is particularly present for the near average prediction individuals, as mentioned above, but also more likely when the characteristic function itself has to be approximated. This approximation entails variance as another potential cause for failure. Now, even if $\mathrm{E}[v(S_1)+v(S_2)] = v(N)$, when $v(N) \approx 0$ there is a high probability of getting a value which is of opposite sign. In fact, only an approximation really close to the desired $v(N)$ will avoid a big scaling error.

To look into the necessity of the scaling step of the algorithm, a new method will be tested in Section 5. DivisiveShapApproxNew works exactly like DivisiveShapApprox, but ignores the scaling. Essentially this means that the Shapley values computed in the leaf coalitions will be the approximated Shapley values of the full game (for Figure 1 this means the approximation is complete when it has reached the green boxes). As long as the decomposition of the game is adequate, this method will be fine, as opposed to the original DivisiveShapApprox which can still run into the aforementioned problems. Avoiding the scaling is however not without its consequences. As the scaling step in DivisiveShapApprox is responsible for keeping the Shapley values on a size where they perfectly sum to the gain in payout from the players, this feature is lost in DivisiveShapApproxNew. Therefore, there is no guarantee that DivisiveShapApproxNew satisfy the Efficiency property of the Shapley values for any more than the leaf coalitions that the Shapley values are based on. This raises questions regarding how one can interpret the approximated Shapley values from this method if the difference in prediction explained by the Shapley values is no way near the actual gain from the players.

4 Machine learning methods

This section presents the two machine learning models which will be used in the experiments in Section 5. First, the choice of machine learning models for this project is explained. Then the linear model is presented in Section 4.1 and the random forest model is presented in Section 4.2.

The linear model is a convenient choice of machine learning model for the control experiments in Section 5.1. As presented in Section 2.3, the linear model has both an exact formula for the characteristic function, $v(S)$, in Equation (1), and an explicit formula for the Shapley values if the features are independent. The explicit formula for the Shapley values can not be used for the purpose of this project, since all experiments in Section 5 will include at least some correlated features. However, the exact formula for v for a linear model with dependent features in Equation (1) is useful as it greatly reduces the computational time from having to use the Monte Carlo integration approach in Equation (6), which would have to be used for other models without an exact formula for the characteristic function. Moreover, having to approximate the characteristic function would introduce variance to the "exact" Shapley values that will be used as the ground truth in the experiments. This is suboptimal, as the error estimates are based on this ground truth. Avoiding this approximation of the characteristic value is therefore favorable.

The random forest model is a machine learning model with typically much higher predictive power than the linear model. The model is much more complex than the linear model and is thus able to represent more convoluted relationships between the features and the output, but it is also more difficult to interpret. The random forest model is chosen as for the experiment in Section 5.3 as it has good predictive power and is a typical black box model for which Shapley values are useful.

4.1 Linear model

The linear model is one of the simplest machine learning models. The model assumes a linear relationship between the observation, y, and the covariates $\boldsymbol{x}$ with an error term, $y = \boldsymbol{x}^\top \boldsymbol{\beta} + \varepsilon$. Note that the vector with the covariates in this representation will usually include a 1 as the first element to allow an intercept in the model, $\boldsymbol{x} = (1, x_1, \cdots, x_n)^\top$. In the experiments in Section 5.1 we do not include an intercept term however, and this element is excluded from the vector. The error term is typically assumed to have a normal distribution, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, and finding the optimal $\boldsymbol{\beta}$ is now the same as solving a least squares problem (Wood, 2015). The fitted linear model is then

$$\boldsymbol{\beta} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{11}$$

where $\boldsymbol{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)^\top$ is the design matrix and $\boldsymbol{y} = (y_1, \cdots, y_k)$ are the corresponding observations.

The prediction from the linear model on an individual with covariates $\boldsymbol{x}$ is

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{\beta}, \tag{12}$$

but several predictions can be made at the same time with matrix multiplication:

$$f(\boldsymbol{X}) = \beta^\top \boldsymbol{X}. \tag{13}$$

4.2 Random forest

The fundamental cornerstone of the random forest model is the decision tree, and thus the decision tree is presented first. Decision trees make their predictions by running the covariates of an individual through a set of comparisons, e.g. if $x_1 < 40$ then follow the tree to the left, otherwise

follow the tree to the right. As this reaches a leaf node, the tree returns a prediction. For classification problems, the leaf node predicts a class, while for regression problems it predicts a value, and is often called a regression tree. In practice this means that regression trees can only produce as many different values as there are leaf nodes.

Decision trees are usually constructed in a greedy top-down manner, called recursive binary splitting (James et al., 2013). For classification trees, each new split is made at the place with the highest information gain (Stuart and Norvig, 2009), which is based on the concept of entropy. Entropy is a measure of disorder, and is defined as

$$H = -\sum_i p_i \log_2 p_i,$$

where p_i is the probability of class i. In this setting, the probability p_i is the share of observations which has reached this part of the tree belonging to class i. The information gain is then the reduction in entropy from a split. Another common option is to use Gini index instead of entropy. The Gini index is defined as

$$G = \sum_i p_i(1 - p_i),$$

and is in fact rather similar in characteristics to the entropy measure.

The regression tree is similarly fitted by minimizing the RSS (residual sum of squares) in each step,

$$\text{RSS} = \sum_{j=1}^{k} \sum_{i \in R_j} (y_i - \hat{y}_j)^2,$$

where $R_1, \cdots, R_k$ refers to the high-dimensional boxes the splits of the tree form and $\hat{y}_j$ is the mean of the observed values y in box R_j (James et al., 2013).

Decision trees can be be grown to perfectly explain most training data, but doing so will lead to too big trees that are likely overfitted to the training data. To avoid overfitting, big decision trees are pruned to make smaller trees. The smaller trees are not only less likely to overfit, but are also more interpretable for humans. The decision trees do not have the same predictive accuracy as many other methods, and are highly sensitive to small changes in the data (James et al., 2013). The predictive accuracy can be considerably improved by creating ensembles of decision trees, which is what the methods bagging, random forest and boosting do.

Bagging and random forest are two closely related methods. They both create an ensemble of decision trees that are made from bootstrapped samples of the training data, instead of the full training data. This results in different trees for each run, and the resulting model is not just one tree, but e.g. 100 different trees. The prediction made by these models is aggregated from the predictions of the different decision trees. For classification problems they select the most common prediction from the decision trees. For the regression problems they predict the average prediction of the decision trees.

The difference between bagging and random forest is in how the splits are made. Bagging produce splits in the exact same way as the single decision tree. If there are some features that have strong predictive power in the data set, they are likely to be in the first nodes of all the different trees, meaning the trees produced by bagging are not very different. A high correlation between the trees fails to lower the variance of the prediction compared to a single decision tree (James et al., 2013). Random forest avoids having correlated trees. When the each split is generated, it is only allowed to consider m different features (typically $m \approx \sqrt{n}$, where n is the total number of features). In this way, the trees are forced to use more of the features when creating the trees, and they are much more diverse than the ones produced by bagging.

Boosting is more complex than bagging and random forest (James et al., 2013), and will in short fit trees on the residuals from previous trees. All the three methods that use ensembles of trees are better than the simple tree. Generally random forest models are better than bagging, while boosting in many cases outperforms even the random forest (Stuart and Norvig, 2009). The main disadvantage of these three models is that the nice interpretable nature of the decision tree is lost when there are many trees that are responsible for the outcome.

5 Experiments

In this section we will compute SHAP values for three differently constructed data sets. First, in Section 5.1 we will use a linear model and multivariate Gaussian distributed data, since we in this case are able to exactly compute the true SHAP values. In Section 5.2, the linear model is replaced by a non-linear model, but the data is still multivariate Gaussian. Finally, in Section 5.3 we use a synthetic data set based on a real data set from DNB. The simulated experiments in Section 5.1 and Section 5.2 are designed to be more controlled and simple variants of the DNB data set. This data set in Section 5.3 contains 28 features, which makes exact computation of the Shapley values quite time consuming. Hence, the simulated experiments in Sections 5.1 and 5.2 will use 15 and 9 covariates, respectively.

In the experiments we will use four different estimators for the characteristic function $v(S)$; `v_indep`, `v_all_multigauss`, `v_cluster_multigauss` and `v_linear`.

For the estimators `v_indep`, `v_all_multigauss` and `v_cluster_multigauss` the characteristic function is estimated using Equation (6). The difference between the three approaches is how they sample the values of the covariates that are not in the coalition S, $\boldsymbol{x}_{\bar{S}}^k$.

`v_indep` follows the approach by Lundberg and Lee (2017) which assumes the covariates in S are independent of the covariates in $\bar{S}$. The covariate values $\boldsymbol{x}_{\bar{S}}^k$ are taken from a random individual for each sample, k. This is the original approach for the KernelSHAP.

`v_all_multigauss` assumes that all the covariates are from a multivariate Gaussian distribution. As explained in Section 2.4, $\boldsymbol{x}_{\bar{S}}^k$ is drawn from a multivariate Gaussian distribution with mean and variance according to the Equations (4) and (5).

`v_cluster_multigauss` assumes that within each cluster (not applicable to KernelSHAP) the covariates are multivariate Gaussian, but that covariates in separate clusters are independent from each other. For the remaining covariates in the same cluster as S, the sampled values is done similarly as in `v_all_multigauss`. For the covariates outside the current cluster, the values are samples from random individuals of the data set as in `v_indep`. This approach is simpler, in the way that it generate less data from the multivariate Gaussian distribution. This could reduce the risk of generating unrealistic data in the case where the covariates are not actually multivariate Gaussian.

The last estimator, `v_linear`, assumes that the model to be explained is a linear model, and is not really an estimator when this is true. The characteristic function is computed using Equation (1), with $\mathrm{E}[\boldsymbol{x}_{\bar{S}}|\boldsymbol{x}_S = \boldsymbol{x}_S^*] = \boldsymbol{\mu}_{\bar{S}|S}$ from Equation (4). This requires only one prediction from the model and is exact, in contrast to the other approaches. `v_linear` is the clear best choice for a linear model when the data is multivariate Gaussian, but should not be applied for any other model.

As explained in Section 3, DivisiveShapApprox assumes $v(\emptyset) = 0$. Thus, for all the four estimators we subtract the mean prediction of the data set (the data set is assumed to be representative for the distribution of the population) from the expected prediction given covariate values $\boldsymbol{x}_S^*$.

5.1 Linear model experiments

In this section we simulate data from a linear model with 15 multivariate Gaussian features. The data sets are constructed to mimic the structure of the DNB data set in Section 5.3, seen in Figure 11. The 15 covariates are divided into 5 groups of 3 covariates each. These groups will be referred to as clusters. Within each cluster, we specify a high correlation, $\rho_{\text{in}} = 0.9$, for all experiments in this section. Between all pairs of two covariates in different clusters we use a lower correlation, ρ_{bw}, which differs between the three different experiments conducted in this section. All covariates are set to have an equal variance of 1. Figures 2, 5 and 6 visualise how these correlation matrices appear in the three experiments. The simulated data sets contain 10000 individuals which are drawn from multivariate Gaussian distributions with covariance matrices as described above and with mean 0.

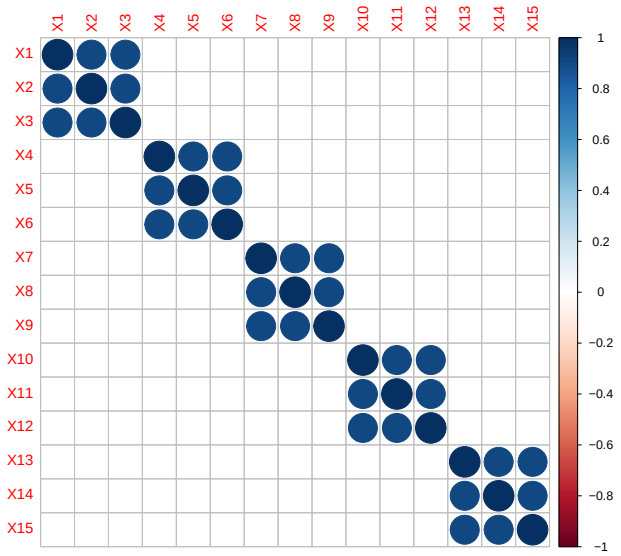


Figure 2: The covariance/correlation matrix used in experiment 1. Within each cluster the correlation is $\rho_{\text{in}} = 0.9$, and between the clusters the correlation is $\rho_{\text{bw}} = 0$, meaning that the clusters are completely independent.

Before looking into the results of concrete experiments, the run times of the KernelSHAP and DivisiveShapApprox were tested on a data set with independent clusters, $\rho_{\text{bw}} = 0$, generated from a linear model with random standard normal distributed coefficients, β from Section 4.1. For a fair comparison both methods used the same estimator for the characteristic function, `v_all_multigauss`. For both approximation methods we tested 9 different values for their tuning parameters, d and β respectively. For each number of samples, d, for the KernelSHAP method, and for each β for the DivisiveShapApprox method, we performed 100 re-runs to get more robust results. For KernelSHAP we used d in the range of 25 to 350 samples. The different values for β were constructed with regards to the maximum size of the leaf clusters allowed by the method. Changing the value of β only have an effect on the DivisiveShapApprox whenever it allows for a different maximum leaf cluster size. The sizes of all the leaf clusters are positive integers, and there is thus only a finite and discrete number of possibilities for tuning of the DivisiveShapApprox method, namely n. The 9 smallest maximum leaf cluster sizes were tested, using $\beta = n^{1/k}$ to allow leaf clusters to be of size up to k. The run times are shown in Figure 3.

From Figure 3 it becomes quite clear that the run time develops differently for KernelSHAP and DivisiveShapApprox. KernelSHAP seems to have a linear run time of $O(d)$, as expected from Section 2.5. DivisiveShapApprox has no theoretical run time when using hierarchical clustering, and appears more complex. The run time appears to increase stepwise at 6 and again at 9, and be constant in between. The reason for this is probably due to the structure of the data set for which the tests were performed. As DivisiveShapApprox aims to partition the data into clusters, it is likely to find the actual clusters of this data set when ρ_{in} is much higher than ρ_{bw}. This means that the partitioning will only ever reach leaf clusters with a multiple of 3 number of features. For example, when DivisiveShapApprox is allowed leaf clusters with up to 5 features, it will still partition clusters of 6 features into two leaf clusters of 3 features, and as a consequence end up being exactly equal to DivisiveShapApprox with maximum leaf clusters of 3. Thus, only when the maximum size of the leaf cluster reaches the next multiple of 3, the method will reach bigger leaf clusters. It will then take longer to compute SHAP values and hence the run time of the method is increased. For maximum leaf cluster sizes of 1 and 2, the same argument does not apply, and the reason the run time decrease here, is because the games are so small that the computation time of the SHAP values in the leaf clusters does not dominate the run time of the method, and joining the SHAP values takes longer.

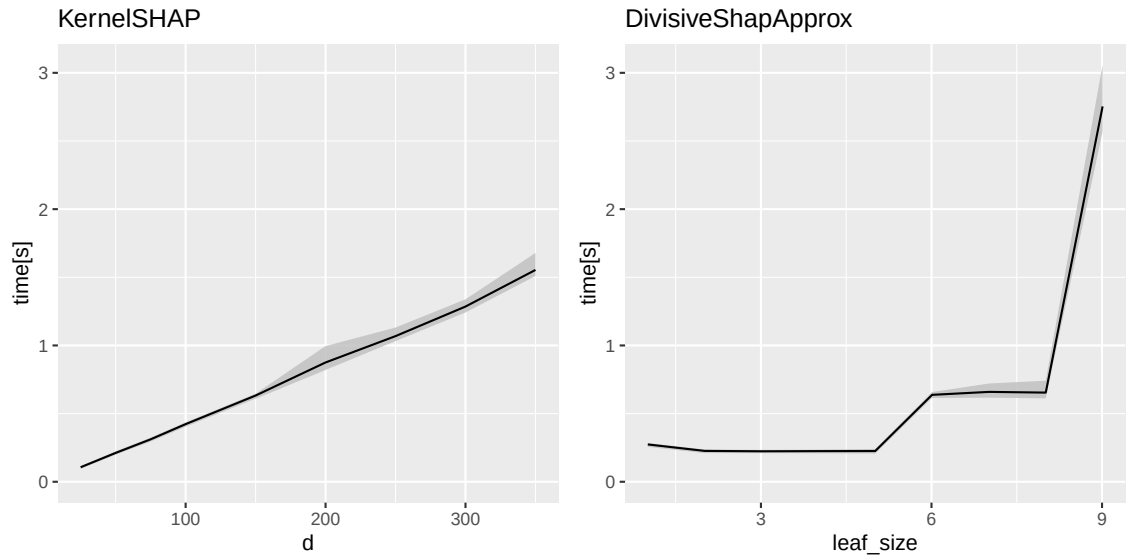


Figure 3: The left plot shows the run time of KernelSHAP with d ranging from 25 to 350 samples. The right plot shows the run time of DivisiveShapApprox with βs corresponding to maximum leaf cluster sizes from 1 to 9. The black lines are the averages of the 100 runs, and the grey area display the center 80% quantile.

5.1.1 Experiment 1

We performed three different experiments with a linear model. The three experiments are referred to as experiment 1, experiment 2 and experiment 3. In the first experiment the between cluster correlation was zero, $\rho_{\text{bw}} = 0$, making the clusters independent (see Figure 2). Using $\beta = n^{1/\sqrt{n}}$ the leaf clusters for DivisiveShapApprox were of max size $3 < \sqrt{15} \approx 3.87$, which is the size of the actual clusters in the simulated data. Thus, with perfect clustering, the partitioning should result in a proper multi-issue decomposition of the characteristic function. The first coefficient of the linear model was set to be zero, $\beta_1 = 0$, while the remaining coefficients were drawn independently from a standard normal distribution.

The SHAP values for one individual in experiment 1 were approximated using KernelSHAP, the original DivisiveShapApprox and the modified version DivisiveShapApproxNew. For the KernelSHAP we used only $d = 50$ samples, so that the computation time was roughly equal for the three methods, see Figure 3. The exact computation of the SHAP values used the actual mean, $\boldsymbol{\mu} = \mathbf{0}$, and covariance matrix as shown in Figure 2, in combination with `v_linear`. The approximation methods were given the mean and covariance estimators of the simulated data, as one would have to do in real applications. To compute values of the characteristic function, v, all three approximation methods used `v_all_multigauss`. Additionally, for this experiment, we also computed the SHAP values using the original KernelSHAP method, i.e. with `v_indep` as estimator for v, and $d = 1000$ samples. The resulting approximations are shown in Figure 4, along with the true SHAP values.

The SHAP values from the original KernelSHAP approach (Lundberg and Lee, 2017), using `v_indep`, in Figure 4 are worse than the SHAP values from the KernelSHAP following the approach by Aas et al. (2019), using `v_all_multigauss`, with MAE of 0.289 and 0.123, respectively. This, despite the fact that the original KernelSHAP was allowed 20 times the number of samples of the other KernelSHAP. The reason for this is because the assumption behind the estimator `v_indep` is far from satisfied, and the resulting values of the characteristic function are far from the true values. Thus, the original KernelSHAP approach will not converge to the correct SHAP values, and will not be able to compete with the other approximation methods. Hence, this approach is excluded from all further experiments.

Figure 4 shows how the original DivisiveShapApprox method can fail in some cases. The

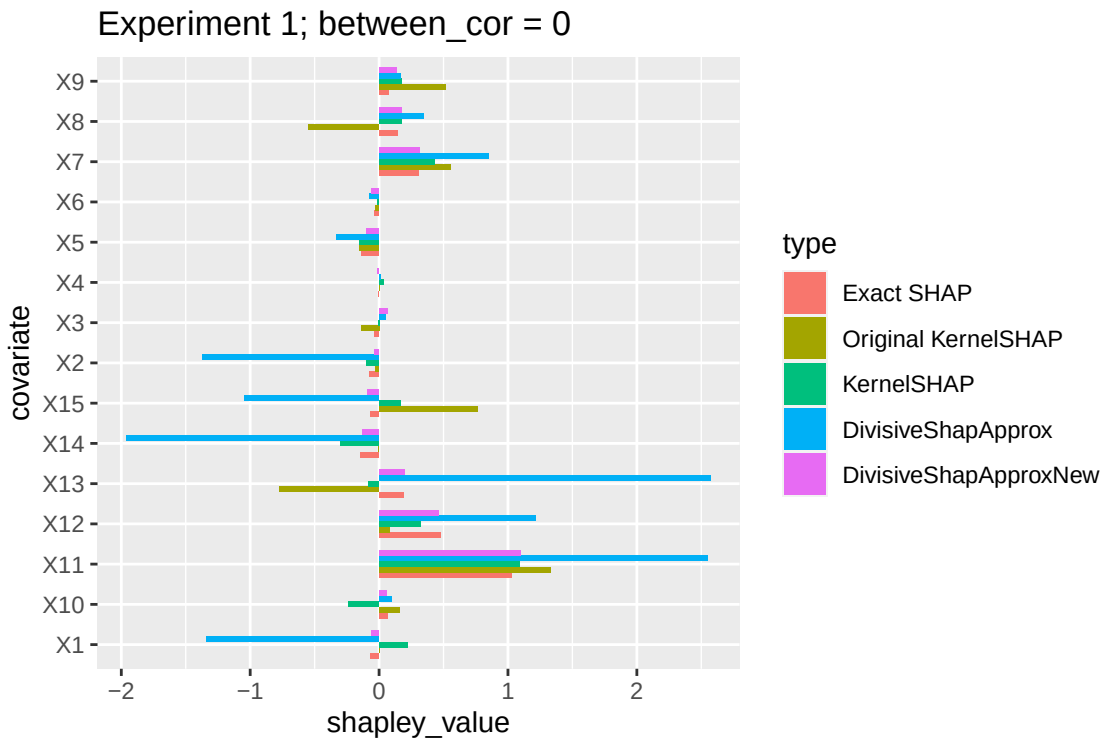


Figure 4: The SHAP values for one individual in experiment 1 with the covariance matrix in Figure 2. Note that the covariates are sorted alphabetically and not by number. The red bars show the exact SHAP values. The olive green bar is the SHAP values from the original version of KernelSHAP, which use `v_indep` as estimator for v, and $d = 1000$ samples. The green, blue and purple bars are the approximations made by KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew, respectively, all using `v_all_multigauss` as estimator for v. KernelSHAP used $d = 50$ samples, and DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/\sqrt{n}}$. MAE of original KernelSHAP = 0.289, MAE of KernelSHAP = 0.123, MAE of DivisiveShapApprox = 0.746, MAE of DivisiveShapApproxNew = 0.0301.

SHAP values for `X1`, `X2`, `X11`, `X12`, `X13`, `X14` and `X15` computed by the DivisiveShapApprox all appear to be scaled too much by the synergy, γ. Looking into the partitioning made by DivisiveShapApprox in this experiment, reveals that the two leaf clusters of covariates `X1` to `X3`, denoted S_1, and covariates `X13` to `X15`, denoted S_5, are joined together. From an exact valuation of these coalitions using `v_linear` we find that $v(S_1) = -0.0944$ and $v(S_5) = -0.0182$, and the joint coalition is valued as $v(S_1 \cup S_5) = -0.1515$. When joining these two coalitions, the computed SHAP values of S_1 and S_5 should be scaled by $-0.1515/(-0.944 - 0.0182) = 1.35$. The total synergy for these leaf coalitions, that is the combined scaling from the start up to the grand coalition, is in fact only 1.25 when the characteristic function is computed exactly. Considering DivisiveShapApproxNew works exactly like DivisiveShapApprox, but without the scaling by synergy rate, we can see from Figure 4 that the actual scaling of the SHAP values in S_5, `X13`, `X14` and `X15` is much higher than 1.25, however, and there has to be more to this explanation.

As mentioned in Section 3.1, DivisiveShapApprox is particularly vulnerable to values of v near 0 for any of the subcoalitions of Figure 1, and Figure 4 is a typical example of this phenomenon. Consider only $v(S_5)$, which is the closest value to 0 here. Over 1000 evaluations by `v_all_multigauss` with $K = 1000$ this estimator had a standard deviation of 0.041, which is more than double of the value itself. In 352 of the 1000 iterations, the estimator returned a positive value, which is the wrong sign and can cause SHAP values as discussed in Section 3.1. Another indication of the scaling of the SHAP values in DivisiveShapApprox being problematic, is that DivisiveShapApproxNew is quite close to the exact SHAP values in Figure 4, raising an important question regarding the necessity of any scaling.

The computation of the SHAP values in Figure 4 was repeated for 100 random individuals. For each individual, the SHAP values were both computed exactly and by the different approximation methods. For each approximation the mean absolute error (MAE) was computed and stored. The average MAE for the approximations by a method is said to be the MAE of the method. Finally, a skill score (Gneiting and Raftery, 2007) is assigned to each method, based on the improvement from a base method. The skill score is defined as

$$\text{Skill} = \frac{\text{MAE(method)} - \text{MAE(base method)}}{\text{MAE(optimal)} - \text{MAE(base method)}} = 1 - \frac{\text{MAE(method)}}{\text{MAE(base method)}},$$

and is the same as used in Aas et al. (2019).

In our case the base method was decided to be the KernelSHAP method with 50 samples, as this is the current most used approximation method with the same computation time as DivisiveShapApprox. This method was compared to the following approaches: KernelSHAP with $d = 100$, 150 and 200 samples, and DivisiveShapApprox and DivisiveShapApproxNew, both with two alternative estimators for the characteristic function, v, namely `v_all_multigauss` and `v_cluster_multigauss`, and with two different values for β to get either 3 covariates in each leaf clusters, $\beta = n^{1/3.1}$ or 6 covariates in each leaf cluster, $\beta = n^{1/6.1}$. The results are presented in Table 1.

Method	v estimator	number of samples	MAE	Skill
KernelSHAP	`v_all_multigauss`	50	0.0928	0
KernelSHAP	`v_all_multigauss`	100	0.0501	0.46
KernelSHAP	`v_all_multigauss`	150	0.0385	0.58
KernelSHAP	`v_all_multigauss`	200	0.0332	0.64
Method	v estimator	leaf cluster size	MAE	Skill
DivisiveShapApprox	`v_all_multigauss`	3	5.08	-54
DivisiveShapApprox	`v_all_multigauss`	6	0.0502	0.46
DivisiveShapApprox	`v_cluster_multigauss`	3	0.105	-0.13
DivisiveShapApprox	`v_cluster_multigauss`	6	0.189	-1.03
DivisiveShapApproxNew	`v_all_multigauss`	3	0.0273	0.71
DivisiveShapApproxNew	`v_all_multigauss`	6	0.0173	0.81
DivisiveShapApproxNew	`v_cluster_multigauss`	3	0.0264	0.72
DivisiveShapApproxNew	`v_cluster_multigauss`	6	0.0153	0.83

Table 1: The MAE and skill score of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over 100 runs of experiment 1, $\rho_{\text{bw}} = 0$. KernelSHAP used $d = 50$, 100, 150 and 200 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$ and $\beta = n^{1/6.1}$ to get maximum leaf cluster sizes of 3 and 6, combined with `v_all_multigauss` and `v_cluster_multigauss`.

From Table 1 one can see that the DivisiveShapApprox is mostly worse than the KernelSHAP method with $d = 50$ samples. The only version to have a positive skill score is one which uses $\beta = n^{1/6.1}$, but this method has a run time comparable to the KernelSHAP with $d = 150$, see Figure 3 (and Table 4), meaning it is still worse than it's KernelSHAP competitor. The DivisiveShapApproxNew methods on the other hand has a skill of about 0.7 with $\beta = n^{1/3.1}$. This score is enough to beat all the included KernelSHAP methods, even though they have up to four times the computation time (Table 4) of the DivisiveShapApproxNew method. One can also note that the two different estimators for v do not affect the MAE of the method significantly, although the two DivisiveShapApproxNew methods which use `v_cluster_multigauss` are slightly better over these 100 runs. This makes sense, as the two methods are close to having the same expectation in this experiment. `v_cluster_multigauss` is slightly more accurate in this experiment however, as it correctly assumes the clusters are independent, while `v_all_multigauss` will use a small correlation between the clusters due to the sample covariance that occurs naturally.

As should be anticipated, the MAE and skill score of the KernelSHAP method improve significantly

as the number of samples, d, increases. There is no pattern in MAE and skill of the DivisiveShapApprox method in this experiment, which is discussed more below. Somewhat surprisingly, the MAE and skill of the DivisiveShapApproxNew also improve significantly as the computation time is increased, going from $\beta = n^{1/3.1}$ to $\beta = n^{1/6.1}$. The reason this may be surprising is that in this experiment, the DivisiveShapApprox and thus also DivisiveShapApproxNew, should be able to get a proper multi-issue decomposition of the characteristic function when having leaf clusters with max 3 covariates, as mentioned above. Thus, one might expect the method to have an equally good decomposition of the game with $\beta = n^{1/3.1}$ as with $\beta = n^{1/6.1}$, yet this does not seem to be the case.

Perhaps the most noteworthy result in Table 1, is the MAE and skill score of DivisiveShapApprox with estimator `v_all_multigauss` and leaf cluster size of 3. To put into perspective, the method's MAE of 5.08 means that the average MAE for the approximations by this method over 100 individuals is 5 times larger than the biggest SHAP value of Figure 4. The skill score of -54 speaks for itself. This result is truly terrible, especially considering that experiment 1 is considered to be the easiest scenario for DivisiveShapApprox. As all MAEs were stored in the making of Table 1, one can discover that the worst SHAP value approximation for an individual had an MAE of 456.691, making it responsible for almost the entire MAE of the method. In fact, after removing only the two worst approximations by this method, the new MAE of the method would be 0.124, and the new skill score -0.34. This goes further to show that the DivisiveShapApprox is very vulnerable to a few extreme mistakes, likely by the same reason as previously discussed, considering DivisiveShapApproxNew does not share this trait.

5.1.2 Experiments 2 and 3

In experiments 2 and 3 we introduce some correlation between the clusters. More specifically, we use a correlation of $\rho_{\text{bw}} = 0.1$ in experiment 2 and a correlation of $\rho_{\text{bw}} = 0.3$ in experiment 3. Figures 5 and 6 show how the correlation between the covariates then appear in the two experiments. These experiments are thought to be less optimal for the DivisiveShapApprox (and DivisiveShapApproxNew) approach, as the subgames are connected, and the multi-decomposition of v might struggle. Particularly, the error of the multi-issue decomposition should increase as ρ_{bw} increases. However, the increasing ρ_{bw} will not ruin the partitioning, and DivisiveShapApprox has no trouble finding the correct clusters from the simulated data sets.

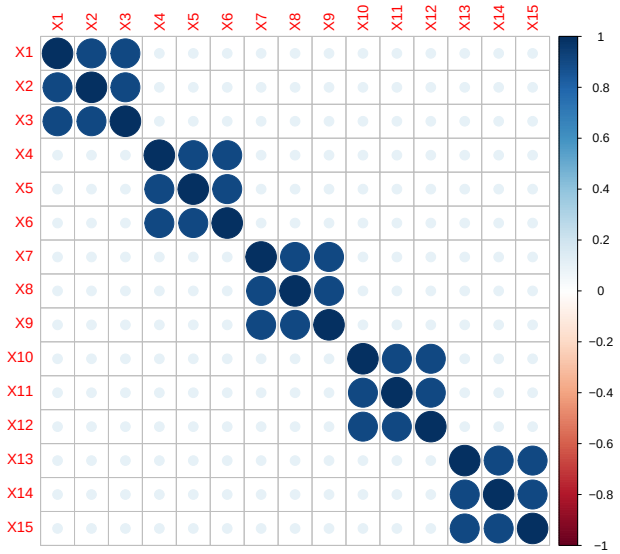


Figure 5: The covariance/correlation matrix used in experiment 2. Within each cluster the correlation is $\rho_{\text{in}} = 0.9$, and between the clusters the correlation is $\rho_{\text{bw}} = 0.1$.

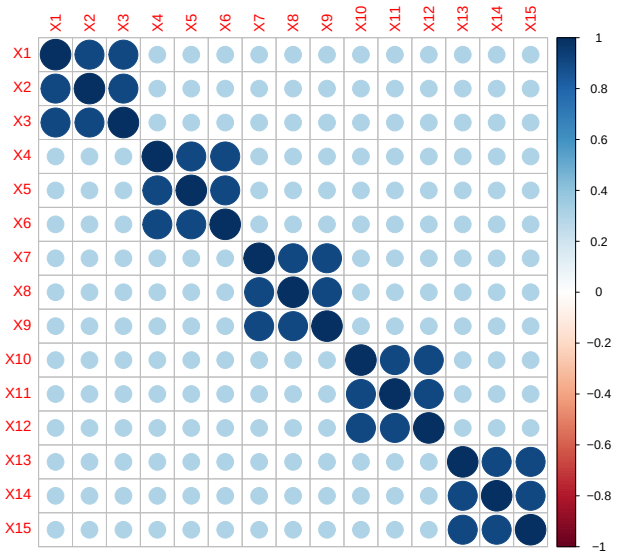


Figure 6: The covariance/correlation matrix used in experiment 3. Within each cluster the correlation is $\rho_{\text{in}} = 0.9$, and between the clusters the correlation is $\rho_{\text{bw}} = 0.3$.

Figures 7 and 8 show the SHAP values for two random individuals in experiments 2 and 3, respectively. Similarly to experiment 1, the true SHAP values were computed as well as the SHAP values from three approximation methods. The three methods were the KernelSHAP with $d = 50$ samples and estimator `v_all_multigauss`, DivisiveShapApprox and DivisiveShapApprox with $\beta = n^{1/\sqrt{n}}$ (max leaf cluster size of 3) and estimator `v_all_multigauss`.

In Figures 7 and 8 the SHAP values computed by DivisiveShapApprox and DivisiveShapApproxNew are better than the SHAP values obtained using the KernelSHAP approach with $d = 50$ samples. The DivisiveShapApproxNew is better than DivisiveShapApprox in both examples, but the MAE of the SHAP values computed by DivisiveShapApprox is much higher than it was in Figure 4.

In experiment 2, Figure 7, we see that the DivisiveShapApprox and DivisiveShapApproxNew correctly identify which covariates that are the four major contributors and the three major inhibitors of the prediction. This is an important task, with respect to the usual applications of the Shapley values. In experiment 3, Figure 8, there are fewer covariates that stand out, but both the DivisiveShapApprox and DivisiveShapApproxNew are able to identify the three covariates that contribute negatively to the prediction, and correctly identify the biggest culprit. For the biggest contributors, DivisiveShapApprox is unable to identify the correct covariates, while DivisiveShapApproxNew correctly identifies the two biggest contributors.

Notice that in Figures 7 and 8, the actual SHAP value for `X1` is not 0, although the coefficient in the linear model for this covariate is 0. This is a consequence of the way SHAP is defined. A covariate can have a non-zero SHAP value solely due to its impact on other covariates, as long as these have an effect on the prediction. I.e. in these experiments, knowing the value of `X1` when `X2` and `X3` are not known, will be very insightful for what these values might be, given the high correlation between these three covariates, and the prediction from the model will take this into account, thus adding to the Shapley values. In fact, in these experiments, as all covariates have some correlation to each other, the value of `X1` always add information regarding the unknown covariates. The intuition behind this could be that if two features are correlated, then in the world of Shapley values they have some sense of symmetry, depending on the grade of correlation, and the effects of one feature should also apply for the other feature. E.g. two perfectly correlated (positively or negatively) features will yield equal SHAP values, due to the symmetry property of the Shapley values.

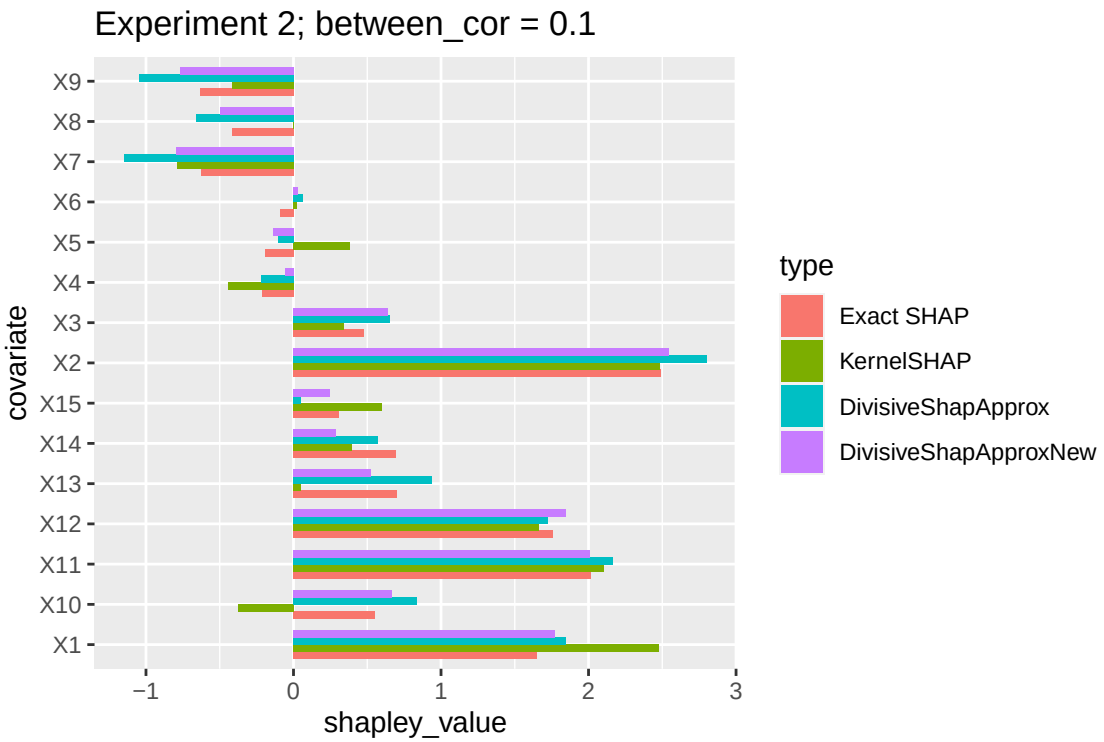


Figure 7: The SHAP values for one individual in experiment 2 with the covariance matrix in Figure 5. Note that the covariates are sorted alphabetically and not by number. The red bars show the exact SHAP values. The green, blue and purple bars are the approximations made by KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew, respectively, all using `v_all_multigauss` as estimator for v. KernelSHAP used $d = 50$ samples, and DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/\sqrt{n}}$. MAE of KernelSHAP = 0.338, MAE of DivisiveShapApprox = 0.212, MAE of DivisiveShapApproxNew = 0.127.

Like experiment 1, experiments 2 and 3 were repeated for 100 random individuals. The resulting MAE of the methods and their skill scores are presented in Tables 2 and 3.

The performances of the methods in Tables 2 and 3 further suggest that DivisiveShapApproxNew is better than DivisiveShapApprox. DivisiveShapApprox is not only always worse than DivisiveShapApproxNew, but it is also often worse than the base method KernelSHAP with $d = 50$ samples, even with leaf clusters up to 6 covariates. The DivisiveShapApprox never outperforms the KernelSHAP with $d = 150$ samples, thus never really making it a reasonable option. Recall however that in both Figure 7 and Figure 8, DivisiveShapApprox is doing better than KernelSHAP with the same computation time. The same can be noted in Table 2 for experiment 2, with estimator `v_cluster_multigauss` and leaf cluster size 3. This shows that DivisiveShapApprox certainly has the potential to make approximations that are better than the popular KernelSHAP method, given equal computation time. Nonetheless, due to the unstable nature of the scaling done in the DivisiveShapApprox method, it is generally worse than its competitors. Previous versions of the performance tests in Tables 1, 2 and 3 which are not reported in this thesis have found quite different skill scores for DivisiveShapApprox in particular, but also to some extent for the other approximation methods. Although they have qualitatively agreed with the results presented in this section, this suggests that 100 test observations is not quite enough to get a completely accurate skill score. However, larger test sets have not been possible in the time frame of this thesis.

DivisiveShapApproxNew is better than the base method in all three experiments, with all skill scores positive in Tables 1, 2 and 3. As the between cluster correlation, ρ_{bw} increases throughout the experiments, the skill score of the method decreases. For experiment 2, with $\rho_{\text{bw}} = 0.1$, the performance of DivisiveShapApproxNew with $\beta = n^{1/6.1}$ is just a little bit better than the

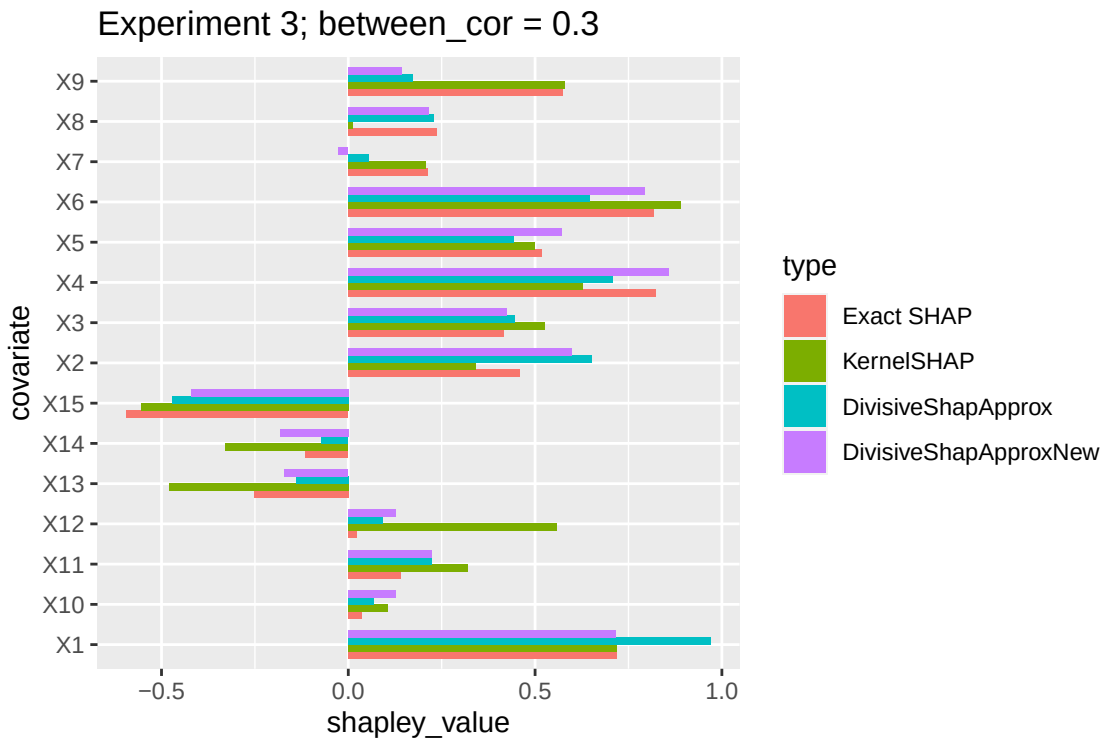


Figure 8: The SHAP values for one individual in experiment 3 with the covariance matrix in Figure 6. Note that the covariates are sorted alphabetically and not by number. The red bars show the exact SHAP values. The green, blue and purple bars are the approximations made by KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew, respectively, all using `v_all_multigauss` as estimator for v. KernelSHAP used $d = 50$ samples, and DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/\sqrt{n}}$. MAE of KernelSHAP = 0.134, MAE of DivisiveShapApprox = 0.124, MAE of DivisiveShapApproxNew = 0.104.

run time equivalent KernelSHAP with $d = 150$ samples, and roughly equal in performance to the just slightly slower KernelSHAP with $d = 200$ samples. For experiment 3, with $\rho_{\text{bw}} = 0.3$, the KernelSHAP with $d = 150$ samples matches the performance of DivisiveShapApproxNew with $\beta = n^{1/6.1}$. If one further increases the between cluster correlation ρ_{bw}, it is expected that this behavior will continue.

A reoccurring event in the different performance tests of Tables 1, 2 and 3 is that the skill score of DivisiveShapApproxNew is higher with the estimator `v_cluster_multigauss` than with the estimator `v_all_multigauss`. The explanation provided in the discussion of Table 1 no longer holds for experiments 2 and 3, where we do have between cluster correlation, yet the difference in skill scores is actually greater. The exact reason for this result is not entirely clear. A possible explanation could be that when `v_cluster_multigauss` samples the covariate values directly from the data set outside the leaf cluster, it actually captures the true distribution of this data better than `v_all_multigauss` does. `v_cluster_multigauss` will guarantee that the distribution of the covariates outside the leaf cluster follows the true distribution, but the correlation between the cluster and the rest of the covariates is ignored. Meanwhile `v_all_multigauss` will use the correlation between the covariates inside the cluster and the rest of the covariates, but it approximates the distribution of the data using the observed mean and observed covariance, which might differ somewhat from the parameters of the true distribution. The results of the performance tests might suggest that the correlation between the covariates in the cluster and the rest is not as important as sampling from the correct distribution outside the cluster.

The run times of the different methods in Tables 1, 2 and 3 were tracked. The differences between the three experiments do not affect the run times of any method. Hence, we compute the average run times over all the three experiments. In Table 4, these run times are presented, including the

Method	v estimator	number of samples	MAE	Skill
KernelSHAP	`v_all_multigauss`	50	0.213	0
KernelSHAP	`v_all_multigauss`	100	0.109	0.49
KernelSHAP	`v_all_multigauss`	150	0.0878	0.59
KernelSHAP	`v_all_multigauss`	200	0.0714	0.67
Method	v estimator	leaf cluster size	MAE	Skill
DivisiveShapApprox	`v_all_multigauss`	3	0.421	-0.97
DivisiveShapApprox	`v_all_multigauss`	6	0.256	-0.20
DivisiveShapApprox	`v_cluster_multigauss`	3	0.165	0.23
DivisiveShapApprox	`v_cluster_multigauss`	6	0.103	0.52
DivisiveShapApproxNew	`v_all_multigauss`	3	0.0912	0.57
DivisiveShapApproxNew	`v_all_multigauss`	6	0.0659	0.69
DivisiveShapApproxNew	`v_cluster_multigauss`	3	0.0743	0.65
DivisiveShapApproxNew	`v_cluster_multigauss`	6	0.0547	0.74

Table 2: The MAE and skill score of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over 100 runs of experiment 2, $\rho_{\text{bw}} = 0.1$. KernelSHAP used $d = 50$, 100, 150 and 200 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$ and $\beta = n^{1/6.1}$ to get maximum leaf cluster sizes of 3 and 6, combined with `v_all_multigauss` and `v_cluster_multigauss`.

computation time of the exact SHAP values.

KernelSHAP has a run time and error trade-off which is discussed in Covert and Lee (2020). The run-time is $O(d)$, which can be seen from Figure 3 and in Table 4, while the variance is $O(\frac{1}{d})$. This is supported by the results in Tables 1, 2 and 3, as the MAE of KernelSHAP steadily decrease as the number of samples, d, increases. The run time and error trade-off for DivisiveShapApprox is less documented, especially in regards to the effect the tuning parameter β has on the error. Tables 1, 2 and 3 reveal that doubling the leaf cluster size in experiment 1, 2 and 3 typically reduce the MAE of the method. This is also shown in Corder and Decker (2019), and explained as less interpolation error. However, as discussed in experiment 1, the error of the original DivisiveShapApprox is at high risk of being subject to extreme scaling, and the different MAEs of this method are mostly just decided by these occasional bad approximations. It is therefore difficult to draw conclusions for DivisiveShapApprox. For DivisiveShapApproxNew the reduction in MAE is more steady, when switching from $\beta = n^{1/3.1}$ to $\beta = n^{1/6.1}$. In experiment 1, the MAE is reduced to about 60%, in experiment 2 the MAE is reduced to about 70% and in experiment 3 the MAE is reduced to about 70% and 60% for the two different estimators. From Table 4, we see that the run times for DivisiveShapApprox and DivisiveShapApproxNew with leaf cluster sizes 3 and 6 are similar to the run times of KernelSHAP with $d = 50$ and 150, respectively. Thus, we can compare the reduction in MAE for DivisiveShapApproxNew to the reduction in MAE for the KernelSHAP when switching from $d = 50$ samples to $d = 150$. In experiment 1 the MAE is reduced to 41%, in experiment 2 the MAE is reduced to about 41% and in experiment 3 the MAE is reduced to about 44%. This means that although the DivisiveShapApproxNew is best with short computation time, as more computation time is granted, the KernelSHAP will eventually become better.

Method	v estimator	number of samples	MAE	Skill
KernelSHAP	`v_all_multigauss`	50	0.199	0
KernelSHAP	`v_all_multigauss`	100	0.112	0.44
KernelSHAP	`v_all_multigauss`	150	0.0878	0.56
KernelSHAP	`v_all_multigauss`	200	0.0717	0.64
Method	v estimator	leaf cluster size	MAE	Skill
DivisiveShapApprox	`v_all_multigauss`	3	0.412	-1.07
DivisiveShapApprox	`v_all_multigauss`	6	0.239	-0.20
DivisiveShapApprox	`v_cluster_multigauss`	3	0.220	-0.11
DivisiveShapApprox	`v_cluster_multigauss`	6	0.174	0.13
DivisiveShapApproxNew	`v_all_multigauss`	3	0.149	0.25
DivisiveShapApproxNew	`v_all_multigauss`	6	0.105	0.47
DivisiveShapApproxNew	`v_cluster_multigauss`	3	0.141	0.29
DivisiveShapApproxNew	`v_cluster_multigauss`	6	0.0848	0.57

Table 3: The MAE and skill score of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over 100 runs of experiment 3, $\rho_{\text{bw}} = 0.3$. KernelSHAP used $d = 50$, 100, 150 and 200 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$ and $\beta = n^{1/6.1}$ to get maximum leaf cluster sizes of 3 and 6, combined with `v_all_multigauss` and `v_cluster_multigauss`.

Method	v estimator	number of samples	run time[s]
Exact	`v_linear`	all (2^{15})	82.878
KernelSHAP	`v_all_multigauss`	50	0.268
KernelSHAP	`v_all_multigauss`	100	0.515
KernelSHAP	`v_all_multigauss`	150	0.774
KernelSHAP	`v_all_multigauss`	200	1.036
Method	v estimator	leaf cluster size	run time[s]
DivisiveShapApprox	`v_all_multigauss`	3	0.265
DivisiveShapApprox	`v_all_multigauss`	6	0.743
DivisiveShapApprox	`v_cluster_multigauss`	3	0.241
DivisiveShapApprox	`v_cluster_multigauss`	6	0.779
DivisiveShapApproxNew	`v_all_multigauss`	3	0.249
DivisiveShapApproxNew	`v_all_multigauss`	6	0.746
DivisiveShapApproxNew	`v_cluster_multigauss`	3	0.231
DivisiveShapApproxNew	`v_cluster_multigauss`	6	0.771

Table 4: The average run time in seconds of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over combined 300 runs of experiment 1, 2 and 3. KernelSHAP used $d = 50$, 100, 150 and 200 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/\sqrt{n}}$ and $\beta = n^{1/6.1}$ to get maximum leaf cluster sizes of 3 and 6, combined with `v_all_multigauss` and `v_cluster_multigauss`.

5.2 Random forest experiments

To bridge the gap between the linear model experiments in Section 5.1 and the experiment with synthetic DNB data in Section 5.3, we also performed two experiments were the data sets were generated from a non-linear model and fitted using random forest models. Due to longer computation time for the predictions from a random forest compared to a linear model, the number of covariates in these experiments was lowered from 15 to 9. The relationship between the output, Y, and the data was specified as follows

$$\eta = \frac{(\text{sign}(x_1(x_2 - 0.2)) + 0.5x_3 + \exp x_4)\cos x_5 + 0.1 \ln |x_6|}{20 + x_7} + 0.1\left(x_8 + x_9^2\right),$$
$$p = \frac{1}{1 + \exp(-\eta)},$$
$$y = H(p - 0.5),$$

where H is the Heaviside function. η may then be interpreted as a latent variable, and p as a probability of belonging to class 1, applying the logistic function to the latent variable η, such that $p \in (0, 1)$. Finally, y assigns the individual to the class 1 if this is the most likely class and to class 0 otherwise.

The data in these experiments also contain 10000 individuals, and were still generated from a multivariate Gaussian distribution such that the handling of dependent features is accurate. The structure of the data is similar to the experiments in Section 5.1, with 3 clusters consisting of 3 covariates each, a high correlation within each cluster, $\rho_{\text{in}} = 0.9$, and a lower correlation between the clusters, ρ_{bw}. The two experiments in this section differ only by the between cluster correlation, and the first and second experiment use $\rho_{\text{bw}} = 0$ and $\rho_{\text{bw}} = 0.2$, respectively.

An exact computation of the SHAP values is no longer possible, since we do not have an exact formula for v for a random forest. However, as previously discussed, v can be estimated by Equation (6), which will be done here. In the "exact" computation of the SHAP values in the experiments of this section, the characteristic function will thus be computed by `v_all_multigauss`, with $K = 10000$. The exact SHAP values and MAE of the different methods presented in this section are therefore not completely exact, but as close as possible given the circumstances.

The random forest models were fitted using the function `ranger` from the R package `ranger`. As input to the `ranger` function we used `probability = TRUE` and `importance = "impurity"`. The first of these inputs tells the function to construct a probability forest as described in Malley et al. (2012) and the second input states that the Gini index (or Gini impurity as it is also called) should be used as the measure of importance when computing the information gain. The random forest models used 500 trees, which is the default for `ranger`. The prediction made by the random forest model is the probabilities of being in a certain class. For classification purposes one can either assign each individual to the class with the highest probability, or in the case with only two classes, assign the individual to class A if the probability for the class is over a certain threshold, and otherwise assign it to class B.

By altering the threshold for classification, one can tune the specificity and sensitivity of the model. The receiver operating characteristic (ROC) curve is a graphical display of this interplay. This curve is created by plotting the sensitivity and 1 - specificity (false positive rate) for all values of the threshold. A naive random classifier will be a diagonal line, and a good classifier will lie above this line and hug the top left corner. The area under the curve (AUC) is a measurement of the performance of a model, and is, as the name suggests, the area under the ROC curve. An AUC of 0.5 is just a random classifier, while anything above 0.5 is better than a random classifier. The best models will have an AUC which is close to 1, which would be a perfect model. Both random forest models in this section have an AUC = 0.992, meaning they have practically been able to recreate the non-linear relationship of the data set.

Like the previous experiments, Figures 9 and 10 show the SHAP values for two individuals estimated with the two random forest models, and Tables 5 and 6 presents the MAE and skill scores

of the different methods. As the random forest experiments are smaller (have fewer covariates) than the linear model experiments, the tests in Tables 5 and 6 include only KernelSHAP with $d = 50$ and 100 samples, and the DivisiveShapApprox and DivisiveShapApproxNew with $\beta = n^{1/3.1}$, i.e. leaf clusters of 3 covariates.

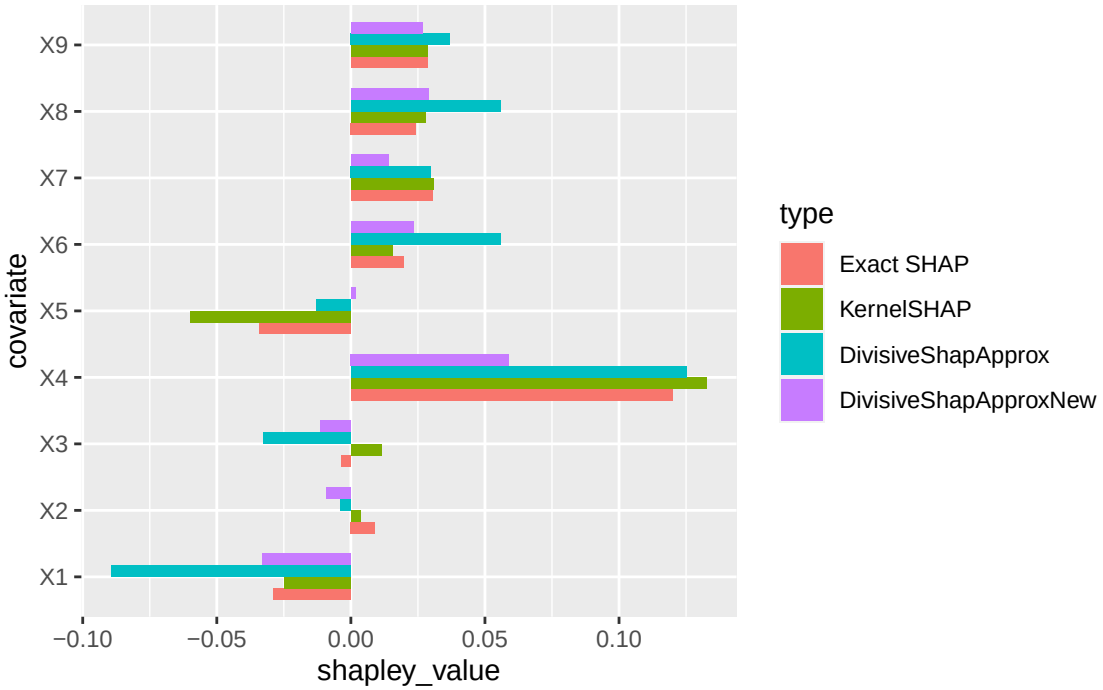


Figure 9: The SHAP values for one individual in the first random forest experiment, with $\rho_{\text{bw}} = 0$. The red bars show the exact SHAP values. The green, blue and purple bars are the approximations made by KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew, respectively. KernelSHAP used $d = 50$ samples, DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$, and all methods used `v_all_multigauss` as estimator for the characteristic function, v. MAE of KernelSHAP = 0.00790, MAE of DivisiveShapApprox = 0.0229, MAE of DivisiveShapApproxNew = 0.0171.

Method	number of samples	MAE	Skill
KernelSHAP	50	0.0119	0
KernelSHAP	100	0.00772	0.35

Method	v estimator	MAE	Skill
DivisiveShapApprox	`v_all_multigauss`	0.190	-15
DivisiveShapApprox	`v_cluster_multigauss`	0.452	-37
DivisiveShapApproxNew	`v_all_multigauss`	0.0217	-0.82
DivisiveShapApproxNew	`v_cluster_multigauss`	0.0217	-0.82

Table 5: The MAE and skill score of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over 100 runs of the first random forest experiment, with $\rho_{\text{bw}} = 0$. KernelSHAP used $d = 50$ and 100 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$ with `v_all_multigauss` and `v_cluster_multigauss`.

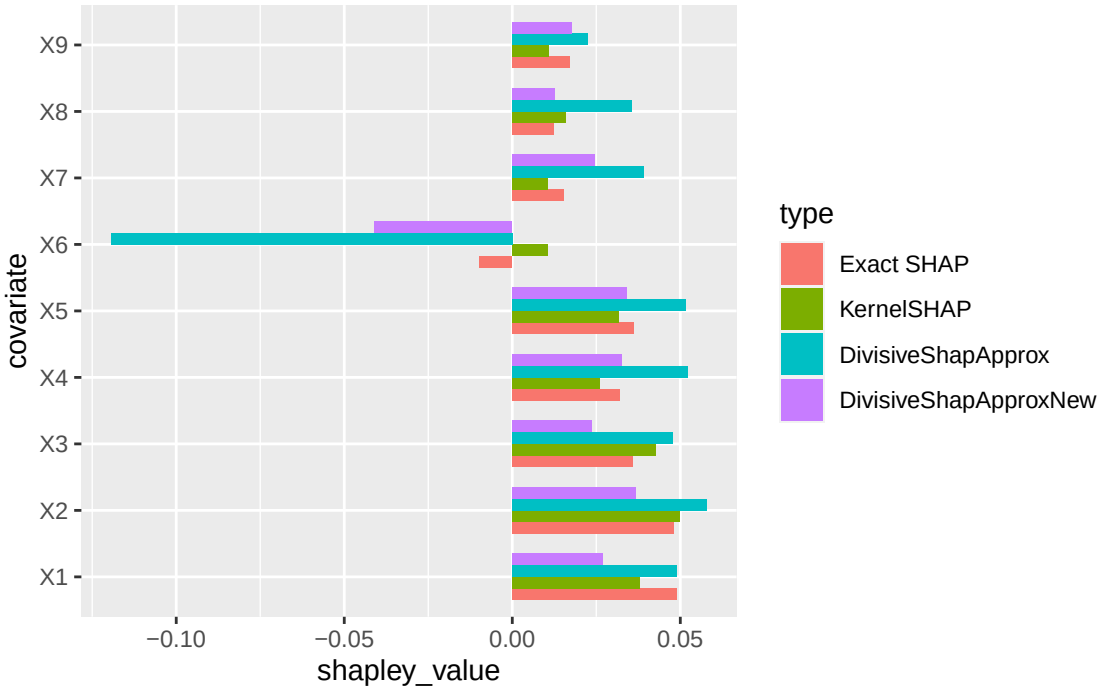


Figure 10: The SHAP values computed in the second random forest experiment, with $\rho_{\text{bw}} = 0.2$. The red bars show the exact SHAP values. The green, blue and purple bars are the approximations made by KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew, respectively. KernelSHAP used $d = 50$ samples, DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$, and all methods used `v_all_multigauss` as estimator for the characteristic function, v. MAE of KernelSHAP = 0.00724, MAE of DivisiveShapApprox = 0.0244, MAE of DivisiveShapApproxNew = 0.00996.

Method	number of samples	MAE	Skill
KernelSHAP	50	0.0147	0
KernelSHAP	100	0.00861	0.41

Method	v estimator	MAE	Skill
DivisiveShapApprox	`v_all_multigauss`	0.151	-9.3
DivisiveShapApprox	`v_cluster_multigauss`	0.0851	-4.8
DivisiveShapApproxNew	`v_all_multigauss`	0.0443	-2.0
DivisiveShapApproxNew	`v_cluster_multigauss`	0.0483	-2.3

Table 6: The MAE and skill score of KernelSHAP, DivisiveShapApprox and DivisiveShapApproxNew over 100 runs of the second random forest experiment, with $\rho_{\text{bw}} = 0.2$. KernelSHAP used $d = 50$ and 100 samples. DivisiveShapApprox and DivisiveShapApproxNew used $\beta = n^{1/3.1}$ with `v_all_multigauss` and `v_cluster_multigauss`.

The MAE in the random forest experiments is generally lower than those in the linear model experiments because the domain of the characteristic function v is more narrow. The resulting SHAP values are thus smaller and so is the MAE. Comparing the bar plots of Figures 9 and 10 to the ones in Figures 7 and 8 it seems as though the relative errors in the random forest experiments are quite similar to those in the linear model experiments in Section 5.1. As can be seen from Figures 9 and 10, the MAE of the SHAP values by the three approximation methods are quite similar. Further, somewhat surprisingly the MAE of the SHAP values by DivisiveShapApproxNew is actually lower in the second experiment, with $\rho_{\text{bw}} = 0.2$, than in the first experiment, with $\rho_{\text{bw}} = 0$.

For the test observations in Figures 9 and 10, the KernelSHAP method with $d = 50$ samples is bet-

ter than both the DivisiveShapApprox and DivisiveShapApproxNew. From Tables 5 and 6 one can see that this is also generally the case, also when using the estimator `v_cluster_multigauss`.

Why does the KernelSHAP method with $d = 50$ samples in this case outperform the DivisiveShapApprox approaches? The only large difference between the experiments of this section and Section 5.1 is the relationship between the explanatory variables and the outcome. While the linear relationships in the experiments of Section 5.1 was easy to decompose, this is not the case for the non-linear relationship in this section. Thus, the multi-issue decomposition of the characteristic function is not particularly successful and the methods that try to partition the variables into clusters do not work as well as in the linear case.

5.3 Synthetic data set from real data

In this section, the methods will be applied on a synthetic data set obtained from the Norwegian financial service group DNB. The synthetic data set is generated from a real data set used for predicting mortgage. To avoid legal or privacy issues associated with sharing real data, DNB has used a method called *synthpop* (Nowok et al., 2016) to generate a synthetic data set from the original data. The original data set is the one used by Kvamme et al. (2018), where each data point contains information about the balances of the checking, savings and credit card accounts and the sum of these. For each of the four balances, we have the mean and standard deviation over a year, the minimum and maximum value over a year and also the standard deviation over the year divided by the mean over the year. In addition to the balances on the accounts, the data contains information on the number of transactions and the amount transferred into the checking account. For both of these we have the mean, standard deviation and maximum over a year as well as the standard deviation divided by the mean. All the features have then been normalized to have mean 0 and standard deviation 1. In total this gives 28 features for each individual along with a response value, `default`, which is either 1 if the individual has defaulted or 0 otherwise. The definition of default used here follows the definition in Basel II.

DNB provided two synthetic data sets, a training set and a test set. The training set contains 12696 individuals (rows) and the test set contains 1921 individuals. As the *synthpop* method used to generate synthetic data is not perfect, the variables in the synthetic data set are not perfectly standardised. From Table 7 we see that the vast majority of the variables have got a mean close to 0, with `sa_std_mean_correct` as the only exception with means of 0.23 and 0.30 in the training and test data sets respectively. For the variances however, there are several variables with variances far from 1. Four of the the six variables displaying standard deviation over the mean (i.e the ones ending in `_std_mean_correct`) have observed variances that are dramatically different from 1, ranging from 0.0039 to 2.47. A last thing to note from Table 7 is that the variances (and the means to some extent) vary between the training set and the test set. For variables such as `ch_min`, `cc_max`, `ch_std_mean_correct` and `sum_std_mean_correct` this difference is quite big, and can potentially be problematic.

Figure 11 shows the correlation between the features in the training data. The correlation in the test data is not shown, but is structurally similar. From the figure it is obvious that there are several clusters in the data sets. The mean, minimum and maximum for each time series are always correlated. As the maximum always will be greater than the other two variables, and the minimum always will be lower than the other two variables, this correlation is natural. The sum is also highly correlated to the savings account. Considering most people will likely have a far larger balance in their savings account than in their checking account or in their credit card account, the sum of the three will thus be mostly dependent on the savings account. The standard deviation of the checking account is also dependent on the amount transferred into the checking account (`in`). Lastly, the standard deviation of the sum is highly correlated to the standard deviation of the checking account (and in turn the `in` variables), due to the fact that for everyday use people tend to use the money in their checking account rather than the credit card or the savings account. Thus the checking account will likely have a much higher standard deviation than the other accounts, and be responsible for most of the variance in the sum. Remember that the variables were standardised before the synthetic data was created, and that the standard deviation of the checking account, `ch_std`, will not appear to have higher variance than the credit card or the savings account in

	Variance		Mean	
	Train	Test	Train	Test
ch_mean	0.76	0.42	-0.0098	-0.024
ch_min	0.78	0.33	-0.0070	-0.028
ch_max	0.94	0.44	-0.0035	-0.022
ch_std	0.94	0.71	-0.0071	0.0044
ch_std_mean_correct	0.070	0.0097	-0.0065	-0.0096
in_mean	0.90	1.35	-0.0083	0.044
in_max	0.74	0.94	-0.010	0.021
in_std	0.90	0.93	-0.0068	0.024
in_std_mean_correct	0.93	0.90	-0.0073	-0.022
cc_mean	0.84	0.65	-0.036	0.0072
cc_min	0.84	0.62	-0.038	0.00094
cc_max	1.14	0.57	-0.028	0.0097
cc_std	0.89	0.81	0.020	0.014
cc_std_mean_correct	0.0039	0.0043	-0.0087	-0.0080
sa_mean	1.07	1.05	0.015	0.038
sa_min	1.18	1.46	0.017	0.044
sa_max	0.99	1.09	0.013	0.040
sa_std	0.59	0.34	0.00038	0.011
sa_std_mean_correct	2.13	2.47	0.23	0.30
sum_mean	0.93	1.00	0.0079	0.032
sum_min	1.07	1.30	0.011	0.041
sum_max	1.11	1.35	0.014	0.045
sum_std	0.96	0.44	0.0023	-0.0035
sum_std_mean_correct	2.33	0.91	0.019	-0.024
tr_mean	1.01	0.99	0.036	0.040
tr_max	1.00	1.09	0.035	0.064
tr_std	1.00	1.08	0.038	0.060
tr_std_mean_correct	0.91	0.89	0.0043	0.013

Table 7: Shows all the variances and means of the features in the two synthetic data sets. In the original data set which the synthetic data set is based on, all the means are 0 and all the variances are 1.

Table 7.

When using the estimators `v_all_multigauss` and `v_cluster_multigauss`, the mean and covariance matrix of the data are needed. Table 7 show that the means and variances are not equal for the two data sets. In order to get the best representation of the population, the two data sets are combined when computing the mean and covariance to be used in the approximation methods.

A random forest model was fitted on this data set using the function `ranger`. The input given to the `ranger` function was the same as in Section 5.2, with `probability = TRUE` and `importance = "impurity"`, and 500 trees. The resulting model has an AUC of 0.855, indicating that the model had acceptable discrimination. It should be noted that on the test data, this random forest model had specificity of 0.996 and a sensitivity of 0.0393, when using a threshold of 0.5. To get a more balanced sensitivity and specificity, the threshold had to be closer to 0.15, yielding a specificity and sensitivity of 0.778 and 0.753, respectively. This does not affect the SHAP values however, as they only use the predicted probabilities.

The SHAP values from the random forest model for the second individual of the test data set were computed. As there is no ground truth in this experiment, it can be quite tricky to assess the performance of the different approximation methods. In order to achieve some sense of ground truth, KernelSHAP was used with $d = 1000$ samples and estimator `v_all_multigauss` (`KernelSHAP 1000`). The SHAP values for this method can not necessarily be considered to be converging to the true SHAP values of this experiment, as they still rely on the assumptions made by the estimator `v_all_multigauss`, namely that the data belongs to a multivariate Gaussian

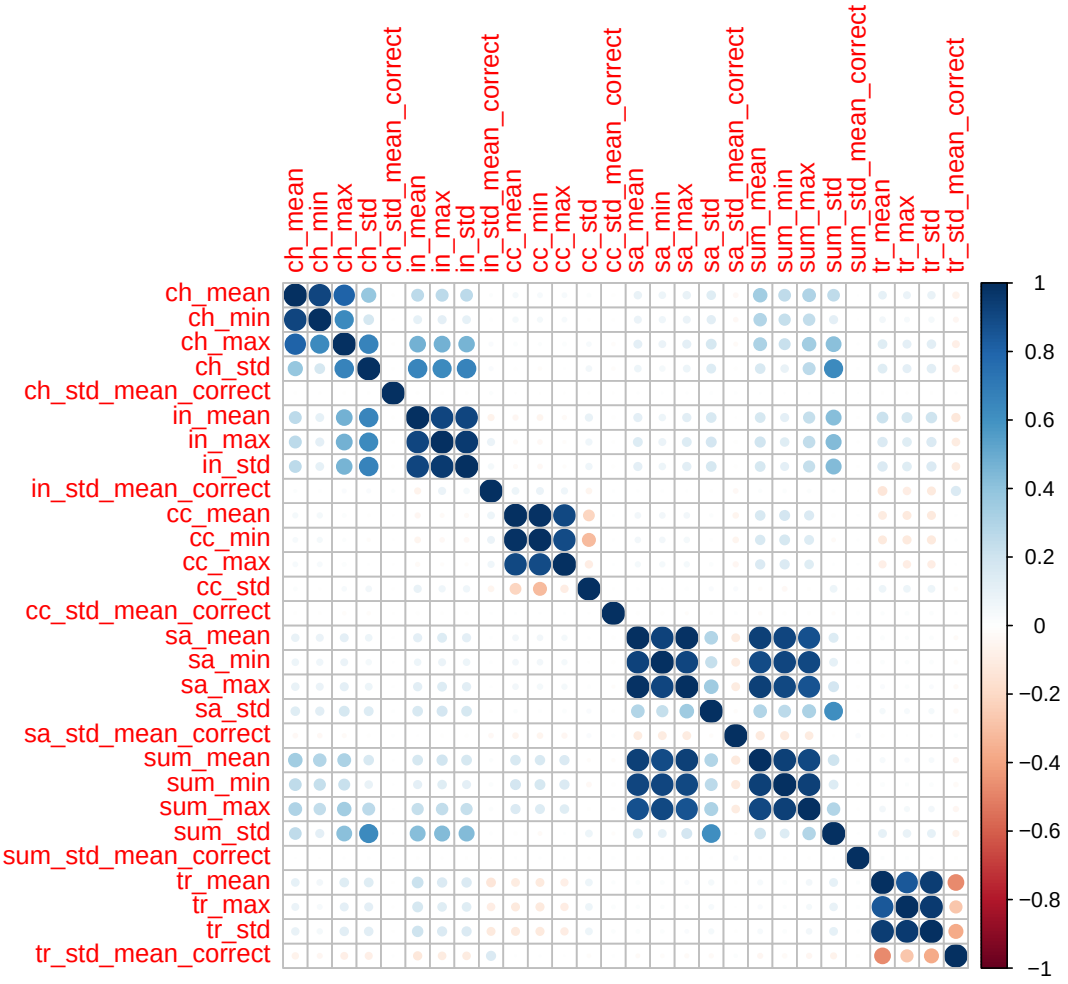


Figure 11: Shows the correlation between the features of the training data set from DNB. `ch` - checking account, `sa` - savings account, `cc` - credit card, `sum` - sum of checking account, savings account and credit card, `in` - amount transferred into checking account, `tr` - number of transactions on the checking account.

distribution. This assumption is however made by all the approximation methods as well, and hence, these SHAP values can be seen as the SHAP values the other approximation methods are trying to achieve. In order to present the approximation by the four different combinations of DivisiveShapApprox and DivisiveShapApproxNew with estimators `v_all_multigauss` and `v_cluster_multigauss`, as well as approximations computed by the KernelSHAP method, the SHAP values are presented in two separate figures, grouped by the sizes of the approximations to make it easier to see each approximation. Both figures show the SHAP values from the KernelSHAP approach with $d = 1000$ samples and `v_all_multigauss`. Figure 12 shows the SHAP values for the following methods: DivisiveShapApprox with `v_all_multigauss` (`DSA v_all_mg`), DivisiveShapApprox with `v_cluster_multigauss` (`DSA v_cluster_mg`), and DivisiveShapApproxNew with `v_cluster_multigauss` (`DSANew v_cluster_mg`). Figure 13 shows the SHAP values for the following approximation methods: KernelSHAP with $d = 100$ samples and `v_all_multigauss` (`KernelSHAP 100`), and DivisiveShapApproxNew with `v_all_multigauss` (`DSA v_all_mg`). All the estimators for the characteristic function v used $K = 10000$ in this experiment, to lower the risk of bad results due to inaccurate estimation of v. The same $\beta = n^{1/\sqrt{n}}$ was used for all DivisiveShapApprox and DivisiveShapApproxNew methods, giving a maximum leaf cluster size of $5 < \sqrt{28} \approx 5.3$.

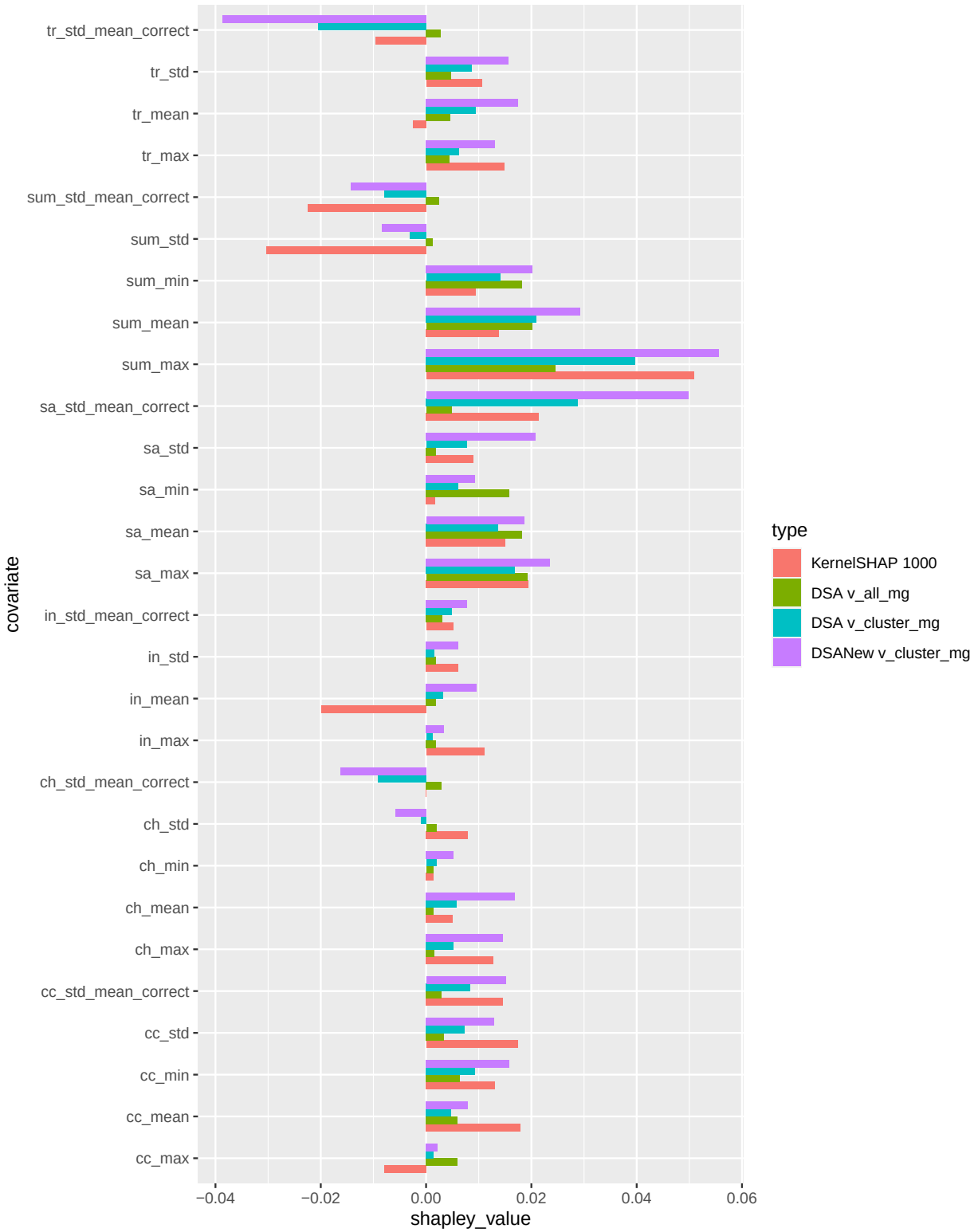


Figure 12: SHAP values of an individual in the DNB test data set for a random forest model. This individual was predicted a probability of default 0.186 higher than the average prediction. `KernelSHAP 1000` is the KernelSHAP method with $d = 1000$ samples and estimator `v_all_multigauss`. `DSA_all_mg` is the DivisiveShapApprox with estimator `v_all_multigauss` and $\beta = n^{1/\sqrt{n}}$. `DSA_cluster_mg` is the DivisiveShapApprox with estimator `v_cluster_multigauss` and $\beta = n^{1/\sqrt{n}}$. `DSANew_cluster_mg` is the DivisiveShapApprox with estimator `v_cluster_multigauss` and $\beta = n^{1/\sqrt{n}}$.

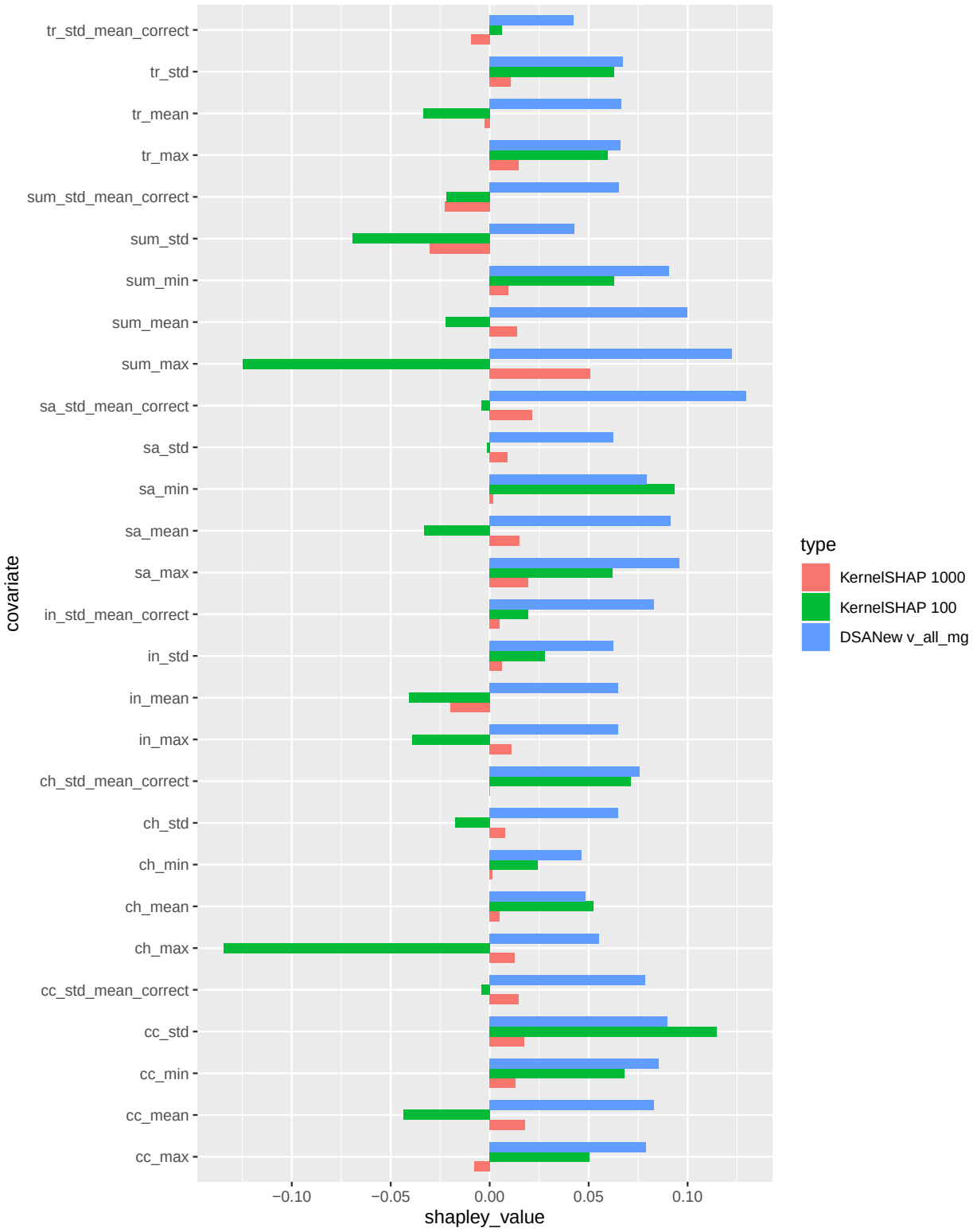


Figure 13: SHAP values of an individual in the DNB test data set for a random forest model. This individual was predicted a probability of default 0.186 higher than the average prediction. `KernelSHAP 1000` is the KernelSHAP method with $d = 1000$ samples and estimator `v_all_multigauss`. `KernelSHAP 100` is the KernelSHAP method with $d = 100$ samples and estimator `v_all_multigauss`. `DSA_all_mg` is the DivisiveShapApprox with estimator `v_all_multigauss` and $\beta = n^{1/\sqrt{n}}$.

KernelSHAP with $d = 1000$ samples is included in both Figures 12 and 13 as a reference. An immediate thing to notice is that the SHAP values from DivisiveShapApprox with both estimators and DivisiveShapApproxNew with `v_cluster_multigauss` in Figure 12 have much more similar sizes to the SHAP values from KernelSHAP with $d = 1000$ samples. All of these four methods agree that `sum_max` is the feature with the largest positive contribution for this individual and three of them agree also for the second largest contributor, `sa_std_mean_correct`. Generally DivisiveShapApproxNew with `v_cluster_multigauss` appears to agree most with the KernelSHAP with $d = 1000$ samples, though there are some features where they do not agree, e.g. `tr_mean`, `in_mean`, `ch_std` and `cc_max` with opposite signs for the SHAP values and a different ranking of the worst contributor. According to DivisiveShapApproxNew it is `tr_std_mean_correct`, while KernelSHAP claims it to be `sum_std`, both being ranked as fourth worst by the other method.

The SHAP values in Figure 13 computed by the KernelSHAP with $d = 100$ samples and DivisiveShapApproxNew with estimator `v_all_multigauss` are not very close to the SHAP values from the KernelSHAP with $d = 1000$ samples. Although both KernelSHAP methods should converge to the same SHAP values, it appears as $d = 100$ samples is far from enough, and the resulting SHAP values from this method do not appear to have any shared qualities to the SHAP from the KernelSHAP with $d = 1000$ samples. The SHAP values from DivisiveShapApproxNew with estimator `v_all_multigauss` drastically overestimate every single feature of the individual. It does however, agree with the KernelSHAP with $d = 1000$ samples for the worst contributor and for the two biggest contributors, but with switched rankings between them. A big concern with the SHAP values computed by DivisiveShapApproxNew with `v_all_multigauss`, is that these are not even close to sum up to difference of 0.186, between the predicted probability of default for the individual of interest and the average prediction. The SHAP values from this method actually sum up to 2.105, which makes no sense as the difference between any two probabilities can never be larger than 1. It is also strange that using the estimator `v_all_multigauss` result in all positive SHAP values for both DivisiveShapApprox and DivisiveShapApproxNew, but not for the KernelSHAP methods. This could have the same potential explanation as discussed in Section 5.2, with `v_cluster_multigauss` using a more true distribution when estimating the characteristic function. Yet, from Figure 11, there are several features having more than 4 strongly correlated features to it, but the leaf clusters by DivisiveShapApprox and DivisiveShapApproxNew could have maximum clusters of 5. This means that there is occasionally a strong correlation between the leaf clusters, and it would seem better to use `v_all_multigauss` to correctly handle this. Another potential explanation could be connected to the assumed multivariate Gaussian distribution of the data, which, if not satisfied, could mean bad estimation of the characteristic function, in particular by the `v_all_multigauss` estimator. The distribution of the data should thus be tested.

One way to compare the distribution of some observations to a known distribution is by using a quantile-quantile plot (Q-Q plot). A Q-Q plot compares the observations to the expected observations from a distribution, by sorting the observations and expected observations and plotting the same quantiles of the two data sets together. If the observations truly belong to the distribution, the points will form a line along the $x = y$ diagonal. If the data points in the Q-Q plot form a line, but not the $x = y$ diagonal, then the data still belong to the distribution bur with other parameters. A shift from the origin implies a relocation of the distribution, and a different slope implies a different scaling of the distribution. If the Q-Q plot does not form a line, this means the observations do not follow the given distribution.

The handling of the dependent features in this thesis rely on the assumption that the data set is from a multivariate Gaussian distribution, meaning that the marginal distribution for all the features of the data set is normal. In the simulated experiments of Sections 5.1 and 5.2 this is the case by design. For the data from DNB this will instead have to be tested. Recall that the synthetic data set was based on a data set with standardised variables, and as such, the assumed distribution of the features in the synthetic data set is the standard normal distribution. Note however that we already know from Table 7 that some of the features have means and standard deviations different from 0 and 1, respectively, but that this is not a problem by itself. However, the Q-Q plots should still produce straight lines if the features are Gaussian distributed. Figures 14, 15 and 16 show the Q-Q plot of three representative features in the synthetic data set, `cc_mean`, `sa_std_mean_correct` and `sum_mean`, respectively.

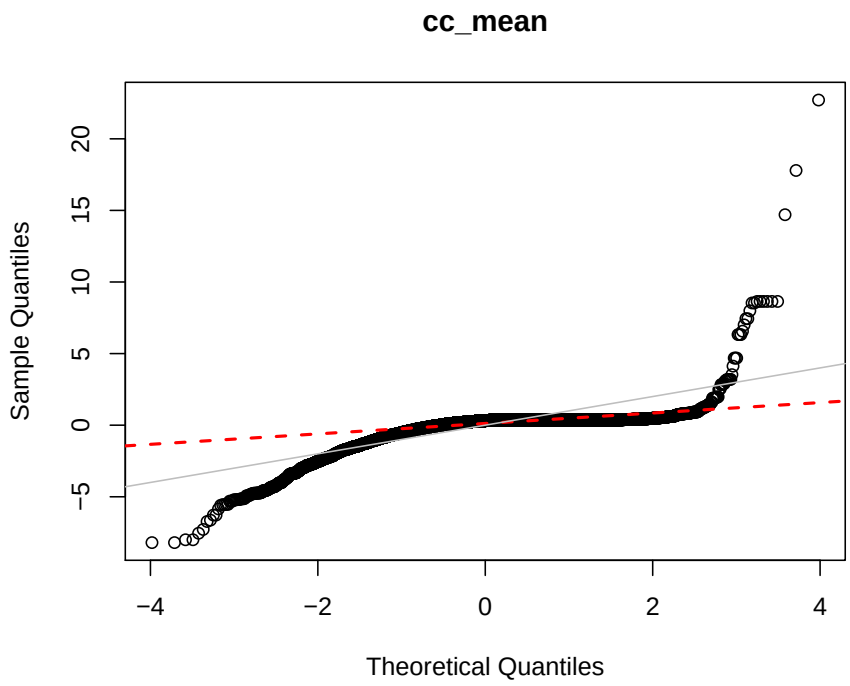


Figure 14: Q-Q plot for the `cc_mean` in the DNB data set. A standard normal distributed variable will follow the gray line. The dashed red line is drawn between the 25th and 75th percentile of the data set.

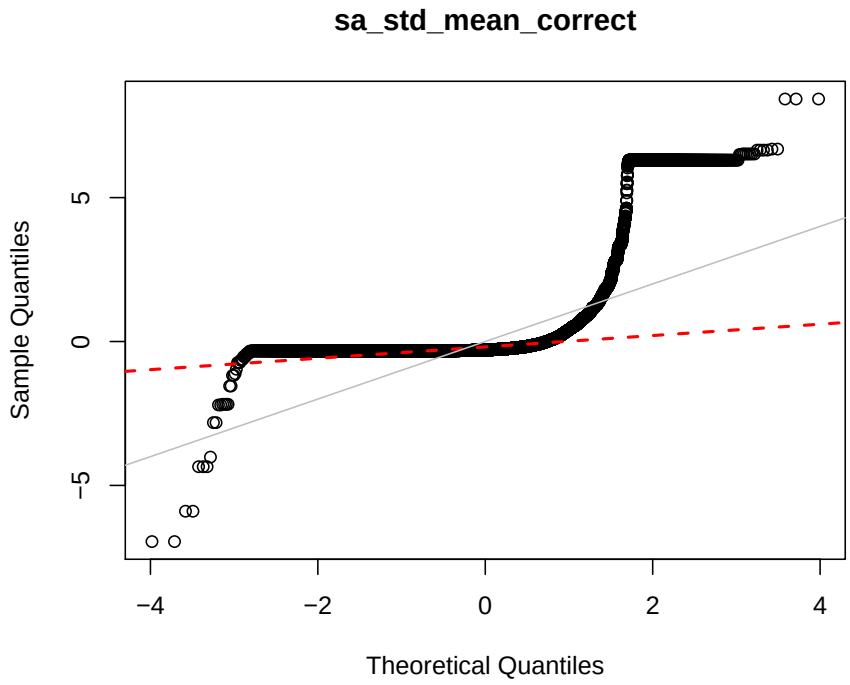


Figure 15: Q-Q plot for the `sa_std_mean_correct` in the DNB data set. A standard normal distributed variable will follow the gray line. The dashed red line is drawn between the 25th and 75th percentile of the data set.

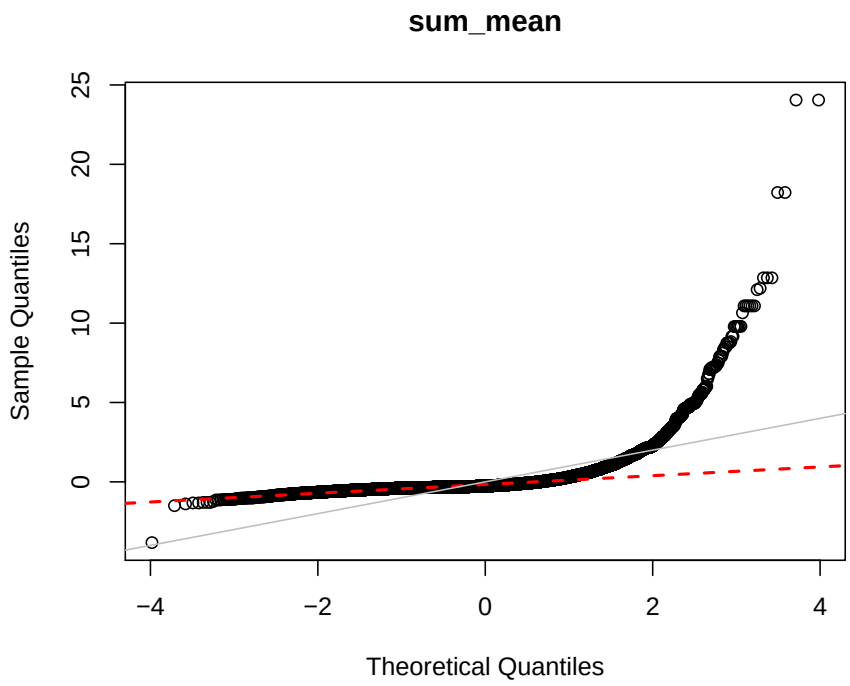


Figure 16: Q-Q plot for the `sum_mean` in the DNB data set. A standard normal distributed variable will follow the gray line. The dashed red line is drawn between the 25th and 75th percentile of the data set.

Neither of the three Q-Q plots in Figures 14, 15 and 16 form a straight line. All the three plots have significant bends, meaning they are certainly not Gaussian distributed. The Q-Q plot of textttcc_mean in Figure 14 are below the dashed red line on the left side and above it on the right side, indicating that the actual distribution of the `cc_mean` is more heavy tailed than the Gaussian distribution. The lack of symmetry implies that the true distribution is skewed. The Q-Q plot of `sa_std_mean_correct` in Figure 15 has a very peculiar shape. This is also heavy tailed and skewed, like `cc_mean`, but also has two plateaus. This implies that `sa_std_mean_correct` has two modes, one being quite far on the right side of the distribution. This does not appear to belong to any known distributions. The Q-Q plot of `sum_mean` in Figure 16 suggest that the `sum_mean` could belong to the exponential distribution, with a slight relocation, as the line dips under 0.

As mentioned, these three features represent all the different shapes of Q-Q plots the features of the DNB data sets have, meaning that not even a single feature of the 28 were actually Gaussian. The exact effect of this unsatisfied assumption is unclear, and the experiments of Section 5.2 cast no light on the subject, as they used only multivariate Gaussian data. A simulated experiment where the data are drawn from other known distributions, but still using the same estimators assuming multivariate Gaussian data could help to get a better understanding of the effects this has. Using other known distributions for the data would still allow exact computation of the conditional distributions, and thus the same quality of testing as was done in Section 5.2.

6 Summary and further work

In this thesis we started by discussing the motivation for the field of interpretable AI, and some of the methods that exist in Section 1. We then presented the theory of the Shapley values, and specifically the SHAP values in Section 2. The two approximations methods of this thesis, the KernelSHAP and the DivisiveShapApprox, were introduced in Sections 2.5 and 3, as well as a modified version of the DivisiveShapApprox method in Section 3.1. In Section 4 we presented the linear model and the random forest model to be used in the experiments in Section 5. The experiments consisted of three simulated experiments with linear models, two simulated experiments with random forest models and non-linear data and finally an experiment with synthetic data from DNB, made from a real data set, also fitted with a random forest model. The results of the experiments were discussed subsequently as the results were presented.

The original DivisiveShapApprox method by Corder and Decker (2019) is an unstable method, and is unsuccessful in outperforming the commonly used KernelSHAP with an equal run time, even for the simplest data set and a linear model. Our modified version of the method, the DivisiveShapApproxNew, performs much better in our experiments. In all experiments with a linear model, this method is either better or equal in performance compared to the KernelSHAP method with equal run time. If a short run time is important, the DivisiveShapApproxNew is the best method here. However, due to different run time and error trade-offs, the KernelSHAP will eventually outperform DivisiveShapApproxNew as one increase the run time of both methods.

The experiments with non-linear data and random forest models saw DivisiveShapApprox and DivisiveShapApproxNew struggle compared to the KernelSHAP approach. Complex relationships of the data make the decomposition of the characteristic function difficult. Also the linear model experiments saw DivisiveShapApproxNew slowly lose its advantage over the KernelSHAP method as the correlation between the features increased. These results suggest the DivisiveShapApproxNew does not work as well when the characteristic function is complex.

In the final experiment, a synthetic data set made from a real data set from DNB was used with a random forest model. The resulting SHAP values were not conclusive in regards to which method was best. Some of the approximated SHAP values appear to be very wrong. The bad results using the estimator `v_all_multigauss` suggest that assuming that the data is multivariate Gaussian is far from the truth. Further testing of DivisiveShapApprox or DivisiveShapApproxNew in complex cases should incorporate other approaches for handling dependent features, e.g. using conditional inference trees as in Redelmeier et al. (2020). Further work could also include more comprehensive testing of the effects that assuming a wrong distribution of the data has for the approximations by the different methods.

Another point of interest which is not addressed much in this thesis is the rankings of the features. Shapley values are often used to communicate which features contribute most and least to a prediction, and the absolute sizes of the Shapley values are sometimes less important. Further work on DivisiveShapApprox or DivisiveShapApproxNew could look deeper into the methods' ability to get the correct rankings of the features, using e.g. Kendall's correlation.

In conclusion, the DivisiveShapApproxNew method seems to thrive when used on simpler models, but should be avoided for more complex models. How complex a model is, however, is not necessarily easy to say up front, and having a method which is only usable for linear models is quite redundant, given their already interpretable nature. KernelSHAP is more versatile, and has the ability to easily be given higher precision by increasing its run time, to the point where it will be better and faster than the DivisiveShapApproxNew. Thus the KernelSHAP is still overall the better choice for approximation method.

References

Aas, Kjersti, Martin Jullum and Anders Løland (2019). 'Explaining individual predictions when features are dependent: More accurate approximations to Shapley values'. In: *arXiv preprint arXiv:1903.10464*.

Aasland, Erik Holst (Jan. 2022). 'A review of the divisive clustering approximation method for Shapley values on the features game and variance game'.

Colini-Baldeschi, Riccardo, Marco Scarsini and Stefano Vaccari (2018). 'Variance allocation and Shapley value'. In: *Methodology and Computing in Applied Probability* 20.3, pp. 919–933.

Corder, Kevin and Keith Decker (2019). 'Shapley Value Approximation with Divisive Clustering'. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, pp. 234–239.

Covert, Ian and Su-In Lee (2020). 'Improving kernelshap: Practical shapley value estimation via linear regression'. In: *arXiv preprint arXiv:2012.01536*.

Doshi-Velez, Finale and Been Kim (2017). 'Towards a rigorous science of interpretable machine learning'. In: *arXiv preprint arXiv:1702.08608*.

Gneiting, Tilmann and Adrian E Raftery (2007). 'Strictly Proper Scoring Rules, Prediction, and Estimation'. In: *Journal of the American Statistical Association* 102.477, pp. 359–378.

Härdle, Wolfgang Karl and Léopold Simar (2019). *Applied multivariate statistical analysis*. Springer Nature.

James, Gareth et al. (2013). *An introduction to statistical learning: with applications in R*. Springer. Chap. 8.

Kvamme, Håvard et al. (2018). 'Predicting mortgage default using convolutional neural networks'. In: *Expert Systems with Applications* 102, pp. 207–217.

Lundberg, Scott M and Su-In Lee (2017). 'A Unified Approach to Interpreting Model Predictions'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc.

Maimon, Oded and Lior Rokach (2005). 'Data mining and knowledge discovery handbook'. In.

Malley, James D et al. (2012). 'Probability machines'. In: *Methods of information in medicine* 51.01, pp. 74–81.

Molnar, Christoph (2019). 'Interpretable Machine Learning. A Guide for Making Black Box Models Explainable'. In: https://christophm.github.io/interpretable-ml-book/. Chap. 6, 8 and 9.

Nowok, Beata, Gillian M Raab and Chris Dibben (2016). 'synthpop: Bespoke creation of synthetic data in R'. In: *Journal of statistical software* 74, pp. 1–26.

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

Shapley, Lloyd S (1953). 'A value for n-person games'. In: *Contributions to the Theory of Games*. Vol. 2. Princeton University Press, pp. 307–317.

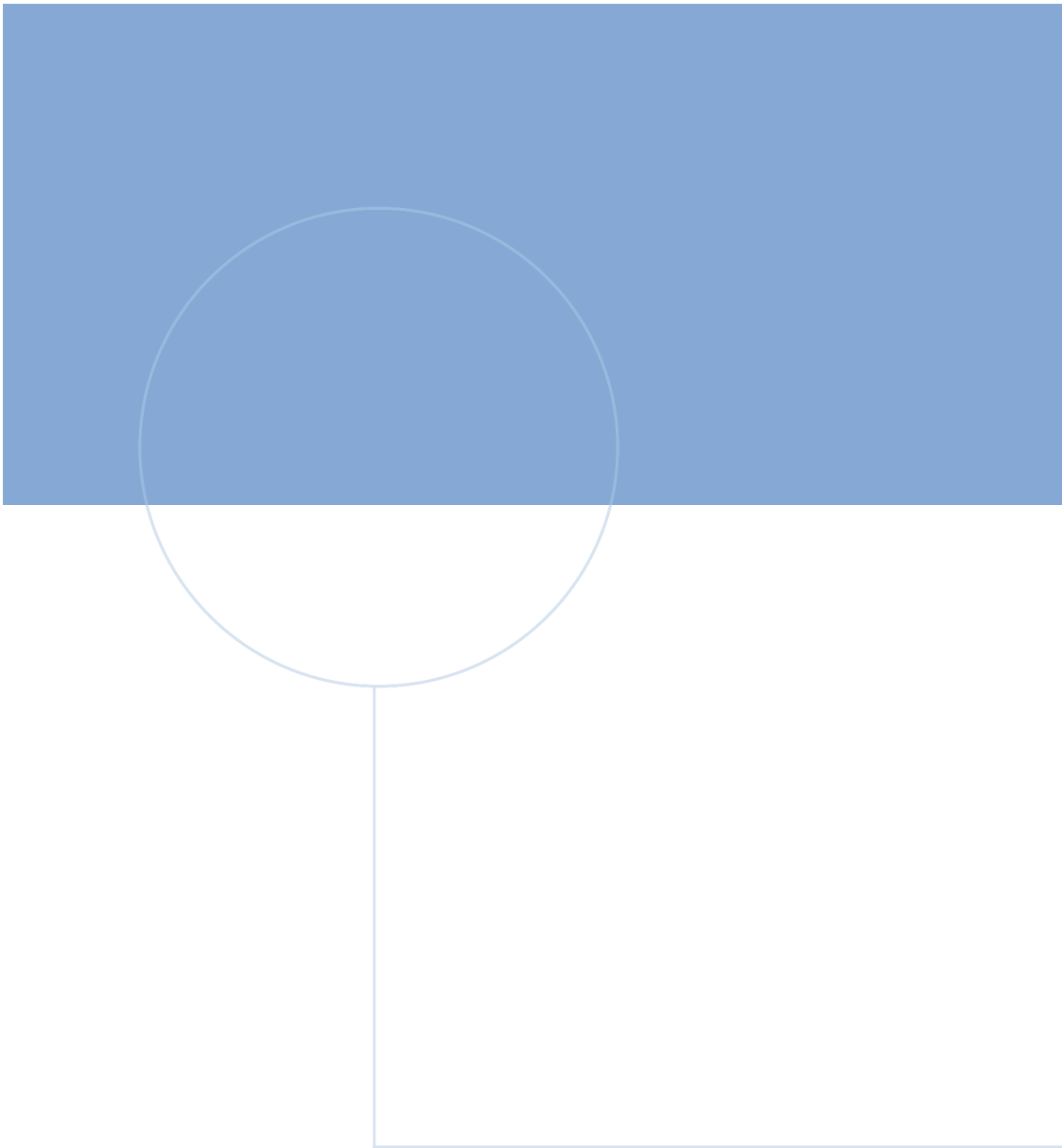
Silver, David et al. (2018). 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play'. In: *Science* 362.6419, pp. 1140–1144.

Soufiani, Hossein Azari et al. (2014). 'Approximating the shapley value via multi-issue decomposition'. In: *Proceedings of the International Foundation for Autonomous Agents and Multiagent Systems*.

Štrumbelj, Erik and Igor Kononenko (2010). 'An efficient explanation of individual classifications using game theory'. In: *The Journal of Machine Learning Research* 11, pp. 1–18.

Stuart, Russell and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach 3rd ed.* Berkeley. Chap. 18.3.

Wood, Simon N (2015). *Core statistics*. 6. Cambridge University Press.



NTNU

Norwegian University of
Science and Technology