Explainable AI via SHAP

by

M.E. Pietersma

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 20, 2024 at 12:30 PM.

Student number:	5331390	
Project duration:	April 22, 2024 – August 20, 2024	
Thesis committee:	Prof. dr. ir. G. Jongbloed,	TU Delft, supervisor
	Dr. N. Budko,	TU Delft

Abstract

As machine learning algorithms become increasingly complex, the need for transparent and interpretable models grows more critical. Shapley values, a local explanation method derived from cooperative game theory, is an explanatory method that describes the feature attribution of machine learning models. This is defined by the contribution a feature has to one single prediction. In this thesis the Shapley value formula is detailed, along with its properties. Three different ways to approximate the Shapley values are then introduced: Monte Carlo method for value function, Monte Carlo method through permutations and Kernel SHAP. Implementation of these methods using different datasets reveals that while both methods produce nearly identical Shapley values, Kernel SHAP is significantly faster. This research contributes to the field by demonstrating the advantages of integrating Shapley values into machine learning models, to enhance model transparency and trustworthiness.

M.E. Pietersma
Delft, July 2024

Contents

1 Introduction

Statistical models are getting more and more complex. They use complicated methods to come to a certain conclusion, which are getting less and less understandable for humans. For example a neural network bases its predictions on connections that are made between data, however these steps can not be seen, so it feels as if the prediction comes out of nowhere. Such a model, with no transparency, is called a black box model. Interpretability is a desirable aspect that scores a certain model on how easily the predictions can be explained. In practice, it often follows that the more accurate a model is, the less interpretable it is. This is because an accurate model, for example a neural network, is often more complex and thus it is less easy to explain the predictions.

An interesting question one could ask is: if a model is accurate why is it not possible to trust the model and why is interpretability needed? There are several reasons why interpretability is essential.

One of the reasons humans need interpretability is to establish trust in a model. In order to feel comfortable in deploying a model in the real-world it helps to see what variables a model bases its decision on, and check that it does not make mistakes in this aspect. Also, if a prediction is made without an explanation, humans are often inclined to make their own assumptions or find their own reasons for why a model makes a specific explanation.

Another reason is that a model could make unnoticed biases. Since a model is built on certain training data, which may already include biases, these can be carried through in the model. For example COMPAS (Northpointe, 2013; Rudin et al., 2020) a machine learning model built to predict the chance that criminals will re-offend, turned out to have a racial bias, meaning that the model gave numerous criminals a higher score based on their race. An explanation method could be able to identify this bias, and thus ensure that the model is changed to not incorporate this bias.

Additionally, interpretability is crucial for regulatory compliance and accountability. In many industries, such as finance, healthcare, and criminal justice, there are legal and ethical standards that require decision-making processes to be transparent and justifiable. In practice, explanation methods enable organizations to demonstrate that their models are making decisions based on valid and lawful criteria, helping to meet regulatory requirements and avoid legal repercussions.

In order to still keep using complex and accurate models but also get explanations for predictions, interpretability methods were developed. One of these methods are the so-called SHAP (Shapley Additive Explanations) values. The goal of these SHAP values is to describe a single instance or prediction by assigning contributions to each of the features that was involved in that prediction.

By using SHAP values, insights can be gained in the inner workings of a machine learning model, how each feature influences specific predictions. This facilitates better trust accountability and fairness in model applications. Furthermore, this also facilitates a better balance between complexity and interpretability, which is crucial for the use of advanced machine learning models in various domains.

The goal of this thesis is to explain the concept of these SHAP values and introduce different ways to calculate these SHAP values and evaluate these different methods. To achieve this goal, first of all, a small background in machine learning is given in Chapter 2. Then, in Chapter 3 the Shapley value formula is introduced, including its properties, and its application to linear regression. Subsequently in Chapter 4, two additional ways to calculate SHAP values are discussed, using machine learning techniques. Finally, in Chapter 5 these different methods will be compared and evaluated on different datasets, and a conclusion will be made on the effectiveness of the different calculation methods.

2 Machine Learning

In theory, at the heart of each statistical model lies a function f on a high dimensional space. This function f can be seen as a representation of the relationship between the inputs, also known as the features, and the outputs, or the target variable. Assuming that $x_1, ..., x_n$ are the individual features, this function can be formulated as:

$$f(\mathbf{x}) = f(x_1, x_2, ..., x_n).$$

To understand this better, imagine one would want to predict the price of a house. Input features could be: size of the house, number of bedrooms, location, proximity to a park etc. Each of these features then represents a dimension in the high dimensional space of f. This function f then takes these features, and produces an output, in this case the predicted house price.

This function f can be estimated through supervised machine learning. The goal of this type of machine learning is to start with a dataset for which the outcome of interest is already known and then learn from this dataset to predict a new outcome. There are two types of supervised machine learning: classification and regression. In classification, the output is a category, for example if a loan is accepted or rejected. Regression has a continuous output, for instance predicting the price of a house based on its features. Mathematically, the difference between the two can be expressed as the function $f : \mathbb{R}^n \to \{1, 2, ..., k\}$ for classification tasks and $f : \mathbb{R}^n \to \mathbb{R}$ for regression tasks. $\mathbb{R}^n$ represents an n-dimensional feature space, with each dimension corresponding to a different feature.

The process of machine learning consists of a number of steps, the first of which is the collection of data. In order to build an accurate model usually the more data one can use, the better. This gives the model enough material to base new predictions on. For example, in the housing price question this would be a dataset of recent house sales and the features of each of these houses.

The next step is to select the type of model that is to be used in the machine learning task. These models can be easily interpretable, such as linear regression or decision trees, or more difficult to interpret such as neural networks [1].

After the model selection, this model is then applied to the data. In practice, this is done by splitting the data into a training set and a test set. The idea behind this is that the training set is used to train the model (i.e. determine the parameters), and the test set is needed to evaluate the model performance on unseen data. This split is necessary, because when building a model based on the whole dataset, this model tends to overfit the data and gives predictions closer to the observed value than one would expect on unseen data. The goal is to find the parameters for the model such that the accuracy of the model applied to the test set is optimal.

Finally, when the model has been learned, it can be applied to a new instance. This instance becomes the input in the machine learning model, and the output is the prediction corresponding to this instance.

Ultimately, machine learning provides a process to accurately discover the function f from the data. By using the power of certain models, it is possible to uncover complex high-dimensional relationships, and thus make accurate predictions.

2.1. California Housing Dataset

As mentioned in the previous section, in order to build a machine learning model an extensive dataset is needed. In this thesis, the California Housing dataset will be used. The machine learning model will aim to predict the median house value of California districts, expressed in hundreds of thousands of dollars ($100,000). This target variable has 8 features, given in table 2.1.

MedInc	Median income in block group
HouseAge	Median house age in block group
AveRooms	Average number of rooms per household
AveBedrms	Average number of bedrooms per household
Population	Block group population
AveOccup	Average number of household members
Latitude	Block group latitude
Longitude	Block group longitude

Table 2.1: Features of California housing dataset

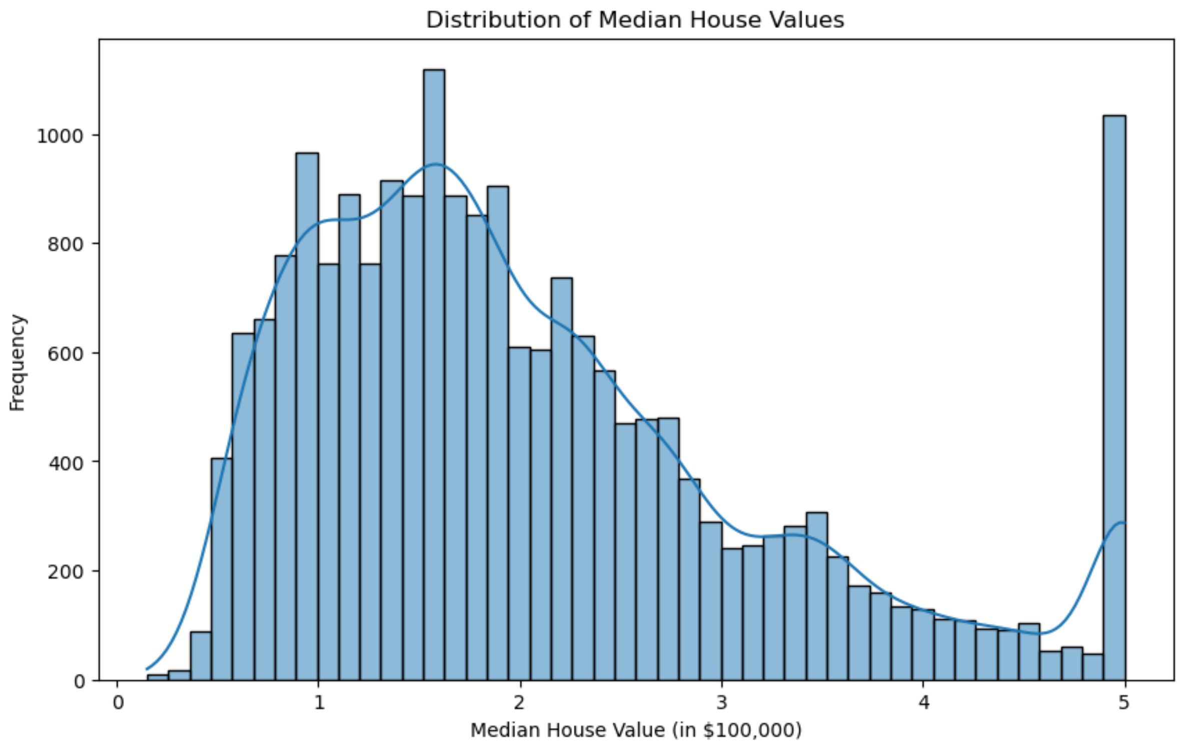


Figure 2.1: Histogram of target variable (20000 houses)

In order to visualize this data a histogram of the target variable is produced in figure 2.1. This histogram shows a long right tail, and an exceptional number of extreme values, making the data hard to model. In 2.2 a correlation heatmap is shown of the features. Especially interesting here is that the feature MedInc has a high correlation with the target variable MedHouseVal. From this, one could assume that the feature MedInc has a high importance in determining the median house value.

As seen in Figure 2.1, the distribution of this data is far from standard, and thus a complex model is chosen to optimally produce predictions. The machine learning model that was chosen to model this

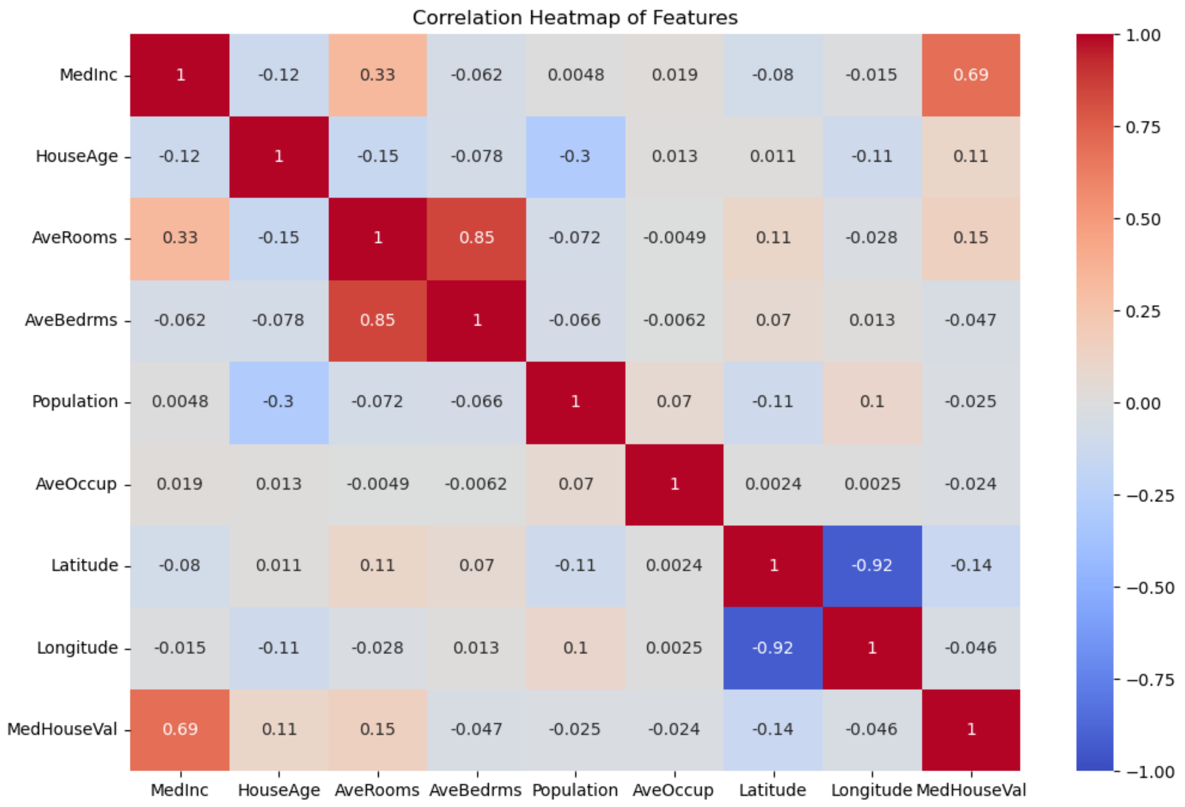


Figure 2.2: Correlation of the features

data is a random forest [2], since this is a relatively fast model, but is not easily interpretable. This is a machine learning technique that constructs an ensemble of decision trees. Each tree is trained on a subset of data, with random subsets of features considered at each split. This promotes diversity among the different trees. The prediction is then procured by averaging the result of all these decision trees. This approach is used to create a more stable prediction and reduce variance in a noisy set.

Furthermore, in order to build a machine learning model, the data has to be split into a training set and a test set.There are multiple theories on the optimal train-test split of data. A commonly used ratio is 80:20%, which draws its justification from the Pareto distribution i.e. that 80% of the outcomes are due to 20% of the causes. However there are multiple papers claiming different optimal splits, in particular, 50-75% for the training set [3] and 50% [4]. Since there does not seem to be agreement on the optimal split, for this dataset a split of 70:30% will be used.

2.2. Explanation methods

The process of machine learning has a wide range of applications, and its strength lies in the machines ability to learn from the data. However, the interpretability of the model can differ a lot depending on the type of model chosen. Namely for linear regression and decision trees it is quite transparent what happens to the data, and which features are the most important. On the contrary for models such as random forests or neural networks, that are based on the premise of mimicking the processes in the human brain, it is not clear what goes on inside the model, a so-called black box model. Since for these models it is hard to see what the predictions are based on, it is important to find a way to also make these models interpretable.

An explanation tries to relate the feature values of an instance to the model prediction in a humanly understandable way. There are certain desirable properties that make a good explanation. Molnar [5] has compiled an extensive list of these properties, here is a small summary:

- Fidelity: the measure of how well the explanation approximates the prediction of the model. This is important because an explanation with low fidelity is does not do a good job in explaining the machine learning model.

- Consistency: the measure of how much the explanation remains unchanged if the model is slightly changed. For example, if two different models are trained, a high consistency would mean that the two explanations are similar.
- Stability: the measure of how much the explanation changes when a similar instance is used. When two instances are almost the same, an explanation is stable if the two explanations are also similar.
- Comprehensibility: how well humans are able to understand the explanations. An example could be how many features are used in the explanation.
- Certainty: Most statistical methods also provide confidence intervals. When a model generates predictions with wide intervals, it indicates that the model is uncertain about the input. This reveals potential weaknesses the model has in handling that specific instance. In contrast, most machine learning models do not create confidence intervals. Thus, an additional desirable property for an explanation method is how well an explanation reflects the certainty the model has in a prediction.

These properties are what categorize a good explanation method. In the next chapter one of these explanation methods is discussed.

3
Shapley values

In chapter 2 it was shown that a machine learning model can be represented by a function f on a high dimensional space $\mathbb{R}^n$. One of the possible explanation methods for this function are the Shapley values. This is a technique used to interpret the model predictions of machine learning model f for individual instances. As seen in Chapter 2, this function f maps inputs **x** $\in \mathbb{R}^n$ to the target variable y. The Shapley values aim to explain how f produces this particular output, for a specific **x**. Thus these explanations do not focus on explaining the whole model. Rather, they zoom in on a single instance and explain how a model has come to that specific prediction. This is called a local explanation method.

The theory behind the Shapley values stems from cooperative game theory. These values were devised to calculate the contribution of each player, to the value of a game in order to distribute a fair payout. This can be applied to machine learning by viewing the features as the players and the value of the game is the output of the machine learning model.

Applying this to a concrete example is done by looking at a prediction for a fixed **x** and trying to explain how each feature value contributes for this specific **x**. The value in this case is the difference between the prediction for the fixed **x** and a sensible prediction in case **x** were unknown. For example in house pricing, a house is predicted to have a value of 300 000, given that the average house price is 310 000. Then the Shapley values would have to explain a difference in price of -10 000, as our predicted house is worth 10 000 less than the expected, or average, house price.

For our fixed **x** we could have the feature values that the house is $50m^2$ which contributes -30 000, that it is an apartment which contributes -20 000, and that it has a park nearby which contributes 40 000, then the total sum is:

$$300000 - 310000 = -30000 - 20000 + 40000 = -10000.$$

This could be one explanation for the predicted value, but how to choose the contributions, as there are many available? One answer to this question was found in 1985 by Lloyd Shapley, who derived the following formula [6]:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(v_x(S \cup \{i\}) - v_x(S)\right)$$

This formula represents the Shapley value, for a fixed $\mathbb{x}$, for one feature, and is given by the weighted average of the contributions to all possible subsets of features. This formula looks quite complex, but it can be broken down into more understandable pieces:

- $v_x(S \cup \{i\}) - v_x(S)$: This term represents the difference between including feature i in a certain subset and not yet including it, in other words the marginal contribution that a feature i has to a subset of features. This is done by applying the previously defined model function f to the subset

S of features, plus feature i, and subtracting this from applying the model to just the features in S. The idea behind the function v_x is to first transform the subset of features to the feature space of all features. Then the machine learning function f is applied. There are multiple ways this transformation can be defined, and two different ways to define this value function will be given in section 3.2.1.

- $\sum_{S\subseteq N\setminus\{i\}}$: This part represents the sum over all over all possible subsets of features (coalitions) S without the feature i.
- $\frac{|S|!(|N|-|S|-1)!}{|N|!}$: This is the weight that each coalition is assigned, this weight is dependent on the number of features in S and is defined by the number of permutations of features that have i on the $|S|$-th place. To ensure that the Shapley values sum up to 1, this weight is then divided by $|N|!$, a proof for this can be found in Appendix A. This implies that the coalitions: $S = \{\emptyset\}$ and $S = N\setminus\{i\}$ get the greatest weight. The idea behind this is that if the subset is small, you can learn a lot about the contribution of a feature in isolation. Furthermore if a subset is big it is possible to learn a lot about the feature specific to the prediction. However if a subset only consists of half of the features, less can be learned about the contribution of the feature, since there are many different subsets consisting of half the features.

3.1. Properties

In Chapter 2 certain desirable properties were mentioned for an explanation method. This section highlights properties that the Shapley values have, and how these properties combined can categorize the Shapley values as 'fair'.

3.1.1. Local Accuracy

One of these properties is the local accuracy property, which coincides with the fidelity property in chapter 2. This property states that the sum of all the Shapley values should be the total difference between the expected value and the predicted value. This means that the explanation should always match the prediction so the sum of the Shapley values for a fixed **x**, should be the difference between the prediction for **x** and the average over all predictions of a dataset. Represented mathematically:

$$\sum_{i=1}^{n} \phi_i = f(\mathbf{x}) - \mathbb{E}[f(X)] = v_x(N)$$

so, expanding the left term, this reads:

$$\sum_{i=1}^{n} \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!}(v_x(S\cup\{i\}) - v_x(S)) = v_x(N)$$

Thus, the goal is to prove that the sum of all the Shapley values is equal to the value of $v_x(N)$. The first step is to write the left side as a difference of two positive sums:

$$\sum_{i=1}^{n} \sum_{S\subseteq |N|\setminus\{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!} v_x(S\cup\{i\}) - \sum_{i=1}^{n} \sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!} v_x(S).$$

Observe that since there is a summation over all Shapley values almost all combinations of features S occur on both the positive term and the negative term. Thus almost all terms cancel each other out due to the symmetry of the positive and negative terms. Only $v_x(N)$ (only on the positive term when $S = N\setminus\{i\}$) and $v_x(\emptyset)$ (only on the negative term when $S = \emptyset$) do not occur on both terms.

Now notice that a certain combination of features S on the positive term appears $|S|$ times with the coefficient:

$$\frac{(|S|-1)!(|N|-|S|)!}{|N|!},$$

(since the length is $|S|-1$). Then observe that on the negative term this combination appears $|N|-|S|$ times with the coefficient:

$$\frac{|S|!(|N|-|S|-1)!}{|N|!}.$$

Now comparing these two terms:

$$\begin{aligned}|S| \cdot \frac{(|S|-1)!(|N|-|S|)!}{|N|!} &= \frac{|S|!(|N|-|S|)!}{|N|!}\\ (|N|-|S|) \cdot \frac{|S|!(|N|-|S|-1)!}{|N|!} &= \frac{|S|!(|N|-|S|)!}{|N|!}.\end{aligned}$$

Since the coefficients are the same on both sides all these terms cancel out, leaving just the contributions of $v(N)$ and $v(\emptyset)$. Next it is possible to discern that:

$$v_x(\emptyset) = \mathbb{E}[X] - \mathbb{E}[X] = 0.$$

Since everything else cancels, all that remains is to calculate the contribution of $v_x(N)$:

$$\sum_{i=1}^{n} \frac{(|N|-1)!}{|N|!} v_x(N) = \sum_{i=1}^{n} \frac{1}{n} v_x(N) = \frac{n}{n} v_x(N) = v_x(N).$$

And thus the Shapley values satisfy the local accuracy property, proving that these values have a high fidelity.

3.1.2. Missingness

The missingness property states that if a feature does not contribute in the prediction, i.e. it does not contribute in any coalition, it should get a Shapley value of 0. Mathematically:

$$\forall S \subseteq N \setminus \{i\} : \ v_x(S \cup \{i\}) = v_x(S) \implies \phi_i = 0$$

This property avoids giving importance to features that do not impact the machine learning function f, which is important for fairness in explanations. This property is a direct consequence from the Shapley value formula, and holds no matter which interpretation of the value function is taken.

3.1.3. Consistency

The consistency property states that if the machine learning function f changes, and thus the value function v_x changes, then if a feature becomes more important in the changed model, its Shapley value should also be higher. This ensures that the Shapley values provide a consistent explanation of the behaviour of a feature, even if the architecture of a model changes.

Represented mathematically: imagine there are two models, with corresponding value functions v and $v*$. If for every subset $S \subseteq N$ holds:

$$v_x(S \cup \{i\}) - v_x(S) \geq v_x^*(S \cup \{i\}) - v_x^*(S)$$

then

$$\phi_i(N, v) \geq \phi_i^*(N, v)$$

This property also follows directly from the Shapley value formula. Thus the Shapley values have three desirable properties. According to Shapley [6], these three properties provide a fair interpretable method for distributing contributions among features.

3.2. Value function

An essential part of the Shapley formula is the value function. As mentioned before, using a machine learning model for only a subset of features is not possible. In order to combat this, the value function v_x contains a transformation of S to the feature space of $\mathbb{R}^n$. After this transformation the machine learning function f can be applied. There are different ways to define this transformation, both of which lead to different definitions of the value function.

First of all, note that in the case that $S = N$, no transformation is needed, and thus this can be directly defined as:

$$v_x(N) = f(x) - \mathbb{E}[f(X)].$$

Note that since the Shapley values are calculated by examining the difference with the expected value, this needs to be subtracted in the value function. Furthermore, in the case where $S = \emptyset$ there are no features, and thus the value function can be defined as:

$$v_x(\{\emptyset\}) = \mathbb{E}[f(X)] - \mathbb{E}[f(X)] = 0.$$

These two values do not use a transformation, and thus are always the same. However, for the other subsets this is not the case.

3.2.1. Value function by distribution

The first way to define the value function, is by integrating over the unknown values by using the distribution of X:

$$v_x(S) = \int f(x_1, x_2, ..., x_n) \, \mathrm{d}\mathbb{P}_{x \notin S} - \mathbb{E}[f(X)].$$

In words, in order to cope with the problem that we only want to insert the actual x-components corresponding to the index set S, the features not in S should be integrated over their distribution to find the partial expected value with respect to those features. Thus the transformation in this case is to integrate f over the distribution for the features not in S.

The problem in the machine learning world is that the distribution of features is unknown or difficult to approximate. So it is not possible to integrate over the distribution of a certain feature, since only the data actually obtained are known.

3.2.2. Monte Carlo Method

Fortunately there exists a way to let computers approximate an integral without having access to the distributions, namely by Monte Carlo integration. This theory can be applied to calculating Shapley values by taking random instances out of the data, since these data are available.

In order to calculate this through Monte Carlo, the machine learning function f, the instance to be explained, a fixed x, and access to the data is needed. From this data a random instance, or row, z is taken. Then, in order to cope with the problem that we only want to insert the actual x-components corresponding to the subset of features S, a combination is made between the fixed x, and the random instance z.

Then to find the value function of a certain subset S we make a combination between the instance to be explained and the random prediction z. Namely for all the features in the coalition S, take the value from x, and all the features not in the coalition S take the value from z. The value function then comes from averaging all the values that come from taking different data points z.

For example assume that a model uses four features for a prediction, and we want to determine the value function of $S = \{1, 3\}$, this gives:

$$\hat{f}^m(\{1, 3\}) = \hat{f}(x_1, z_2, x_3, z_4),$$

where each m is given by a different random data instance z. Now we take M different instances from the data z, and calculate the average to find the value function for $S = \{1, 3\}$:

$$v(\{1,3\}) = \frac{1}{M}\sum_{m=1}^{M} \hat{f}^m(\{1,3\}) - \mathbb{E}[f(X)].$$

Note that, as before, the expected value is subtracted from this average. This is because the Shapley values attempt to explain the feature importance by looking at the difference between the predicted output and the expected output, so this always needs to be subtracted. It may seem logical to also approximate this term $\mathbb{E}[f(X)]$ by an average, however this is unnecessary since to calculate Shapley values, two value functions are subtracted from each other. This causes the expectation term to cancel out.

With this definition the first implementation in Python of the Shapley values can be given. Using the data from Section 2.1, the Shapley values can be used to analyse the predictions of a random forest model predicting the median house value of a block.

Figure 3.1 represents a waterfall plot of the Shapley values for a certain instance where the predicted median house value is $274,300, and the expected house price is $207,200. The bottom of a waterfall plot starts as the expected value of the model output, this is approximated by the average of all predicted house prices. Then each row shows how the negative (blue) or positive (red) contribution of each feature moves the value from the expected model output to the model output for this instance. On the left side the features are given with the corresponding feature values for this instance x. These Shapley values have been calculated based on 100 different random data instances z per coalition, so in this case $M = 100$.

The biggest result of this waterfall plot is that MedInc seems to be the most important feature, after that Latitude and Longitude also play a, albeit a lot smaller, role. Here Longitude affects the prediction positively and Latitude negatively according to the calculated Shapley values. The other features seem less important for this prediction, according to this explanation.

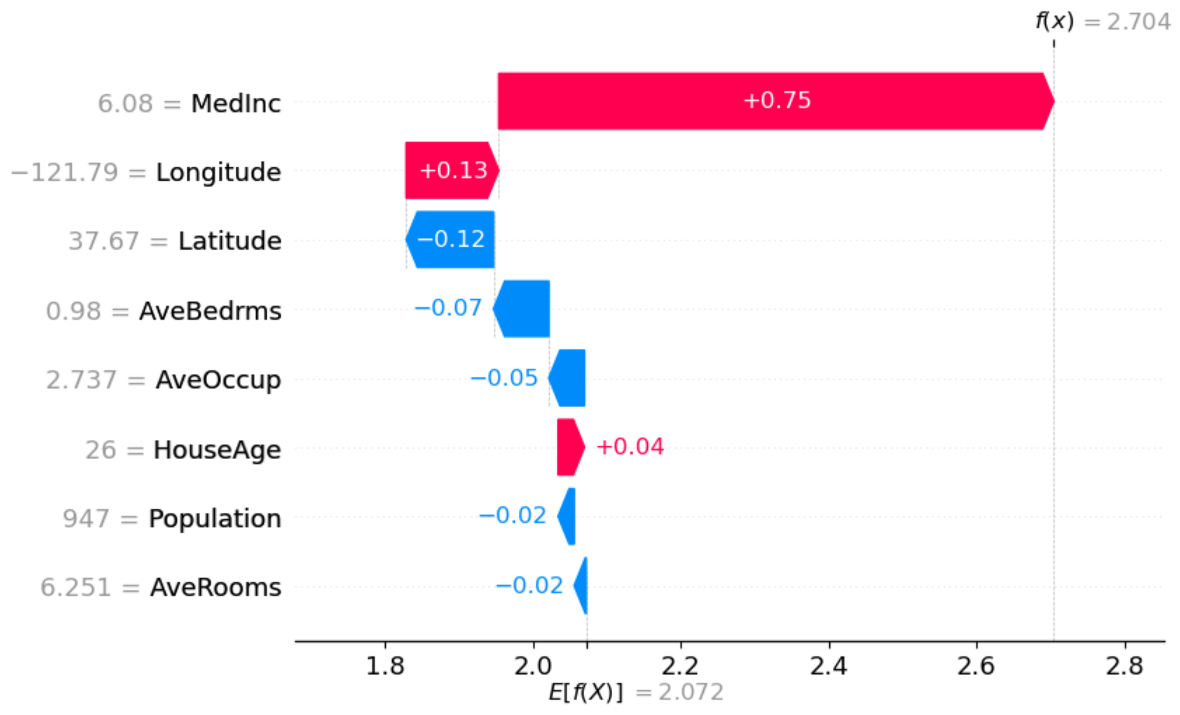


Figure 3.1: Waterfall plot of Shapley values using Monte Carlo Method

3.3. Shapley values for Linear Regression

SHAP is a model-agnostic method, meaning it can be applied to any type of machine learning model. In order to get a better feeling for the Shapley values, the first step is to show how these values are calculated using an interpretable model, namely linear regression. Given the function for linear regression:

$$y = b_0 + b_1 x_1 + \dots + b_n x_n$$

how can the Shapley value formula:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} (v_x(S \cup \{i\}) - v_x(S))$$

be applied to this function?

To give a feeling how this works, it is possible to zoom in on the marginal contribution: $v(S \cup \{i\}) - v(S)$. Take for example $S = \emptyset$, then this difference becomes:

$$\begin{aligned} v_x(\{i\}) - v_x(\{\emptyset\}) &= b_0 + b_1 \mathbb{E}[X_1] + \dots + b_i x_i + \dots + \mathbb{E}[X_n] \\ &\quad - (b_0 + b_1 \mathbb{E}[X_1] + \dots + b_i \mathbb{E}[X_i] + \dots + b_n \mathbb{E}[X_n]) \\ &= b_i (x_i - \mathbb{E}[X_i]) \end{aligned} \tag{3.1}$$

Thus the marginal contribution in this case is $b_i(x_i - \mathbb{E}[X_i])$. Now note that since the linear regression equation is linear, the marginal contribution for an arbitrary set S becomes:

$$v_x(S \cup \{i\}) - v_x(S) = v_x(S) + v_x(\{i\}) - v_x(S) = v_x(\{i\}) - v_x(\{\emptyset\}),$$

since as seen in section 3.2.1 $v_x(\{\emptyset\})$ and thus the difference seen in equation 3.1 holds for every set S. Thus the Shapley formula becomes:

$$\begin{aligned} \phi_i &= \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} b_i (x_i - \mathbb{E}[X_i]) \\ &= b_i (x_i - \mathbb{E}[X_i]) \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \\ &= b_i (x_i - \mathbb{E}[X_i]). \end{aligned}$$

Proof that the sum: $\sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!}$ is indeed 1, can be found in Appendix A.

Consequently for linear regression a quick way is found to calculate the Shapley values for each feature. However, this only works because the model is linear but for other models, different techniques need to be used to find the Shapley values.

4 SHAP

In chapter 3 the formula for Shapley values was introduced, with certain properties. SHAP (Shapley Additive Explanations) focuses on the application of Shapley values to machine learning. To be clear, there is no big difference to SHAP and Shapley values, SHAP just brings new and more modern techniques in order to calculate the Shapley values quicker. In this chapter two different methods are introduced that were developed specifically to calculate Shapley values.

4.1. Monte Carlo Method by Permutation

As seen in section 3.2.2, the value function can be approximated through a Monte Carlo method. However, as the number of features increases, the number of possible subsets also increases exponentially. This leads to longer running times to calculate all the value functions for each subset, increasing exponentially with each additional feature. In order to combat this a second type of Monte Carlo estimation can be used.

One could assume that in order to utilize less coalitions, it is possible to take random subsets of features. However, in practice, then applying the Shapley weights to these random subsets is not trivial, since one of the useful identities of using all possible coalitions is that the weights sum up to 1. By picking a limited amount of coalitions, this identity is lost and the requirements of Shapley values are not fulfilled.

A different way to approximate the Shapley values through Monte Carlo estimation was defined by Strumbelj et al.[7], using permutations. Going back to the game theoretical definition of Shapley values, these weights can be derived by looking at all possible permutations. Then coalitions can be formed by looking at a specific feature in this permutation, and including all features before this feature in the coalition, and the other features are not in the coalition. Since in a coalition the order does not matter, some coalitions occur more often than others, and this distribution is the same as with the Shapley weights.

The Monte Carlo process of finding the Shapley value of feature i of instance x, given the machine learning model f and the number of iterations M, is applied using the following steps:

1. Draw a random instance, or row, z from the data.
2. With N the number of features, choose a random permutation o, and order both x and z accordingly:
   - $x_0 = (x_1, ..., x_i, ..., x_N)$
   - $z_0 = (z_1, ..., z_i, ..., z_N)$

   Thus the coalition consists of the features x_1 until x_i.
3. Construct two new instances, this is the same principle as the Monte Carlo estimation for the value function, for every feature in the coalition keep the value of x, and every feature not in the

coalition use the value of z. Since the goal is to find the contribution of feature i, construct two new instances, one with the value of x for feature i, and the other without:

- $x_{+i} = (x_1, ..., x_{i-1}, x_i, z_{i+1}, ..., z_N)$
- $x_{-i} = (x_1, ..., x_{i-1}, z_i, z_{i+1}, ..., z_N)$

4. Reorder the two instances according to the original feature order, apply the machine learning function to the two instances, and find the marginal contribution by subtracting them:

$$\phi_i^m = f(x_{+i}) - f(x_{-i})$$

5. Approximate the Shapley value by computing the average over all marginal contributions:

$$\phi_i = \frac{1}{M} \sum_{m=1}^{M} \phi_i^m$$

The biggest difference between this method and the previously explained Monte Carlo method in Section 3.2.2, is that this one does not need all subsets of features, and thus defines a more efficient way of calculating Shapley values.

Using the same instance as in Section 3.2.2, using this Monte Carlo algorithm yields the following Shapley values in Figure 4.1.

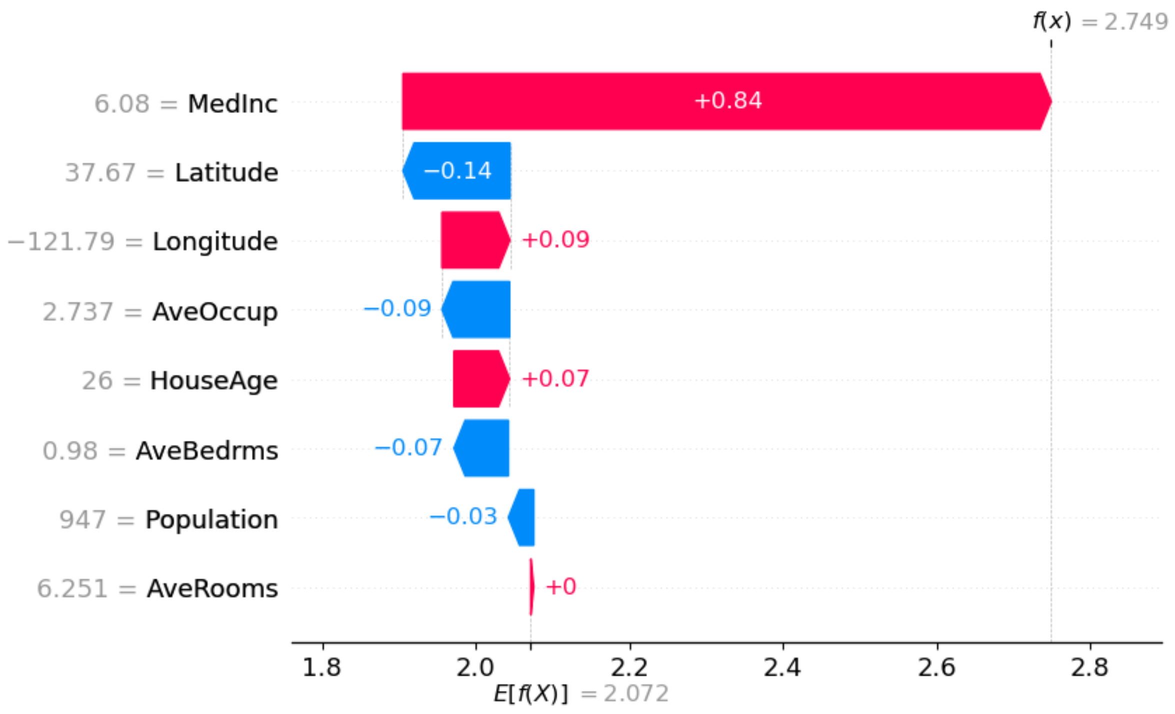


Figure 4.1: Waterfall plot of Shapley values using second Monte Carlo Method.

While it has slightly different values, the waterfall plot in Figure 4.1, does seem mostly similar to the one created in Figure 3.1. For example the most important feature by far is still MedInc, and after that a lot less important are Latitude and Longitude. The differences are mostly in the less important features, according to this calculation of Shapley values. These Shapley values have been calculated using 10,000 total iterations.

4.2. Kernel SHAP

This thesis has already detailed that Shapley values can be approximated via two different Monte Carlo methods. However, Shapley values can also be approximated by using different machine learning techniques, one of these techniques results in Kernel SHAP. This method combines the Shapley values theory with linear regression.

4.2.1. LIME

In order to further explain Kernel SHAP, first we take a step back and look at a very similar method in Machine Learning: LIME (Local Interpretable Model-Agnostic Explanations) [8]. The idea behind LIME is to create an interpretable model to explain how a non-interpretable model comes to each explanation x. This is a version of a surrogate model, which is a model that uses interpretable models to approximate black box models. LIME does this locally, thus just like the Shapley values, it creates a different surrogate model for each explanation of the data. Examples of interpretable models could be linear regression, or decision trees.

LIME attempts to find these surrogate models by looking at how differences in data affect the prediction. The method behind this is: start only with the instance x to be predicted, and the black box function f. Then it is possible to input different data points into the black box model, and LIME tests what happens when variations of data are given into this model. This creates a new dataset, with different perturbations of data leading to different predictions. Then each perturbation is weighted, based on the distance between this perturbation, and the instance to be explained x. Then, based on this new dataset a weighted interpretable model can be trained. The final step is to explain the prediction by interpreting the model.

Mathematically this model is created by minimizing:

$$\hat{g}(x) = \text{arg min}_{g \in \mathcal{G}} L(f, g, \pi_x) + \Omega(g).$$

As seen by this formula the goal of LIME is to find the explanation function g, this g represents an interpretable model, for example a linear regression model or decision tree. $\mathcal{G}$ is a collection of all interpretable models. The goal of the explanation function g is to minimize the loss, which measures how close the explanation of g is to the prediction from the black box model. f is the black box model, and π_x is the weight given to each variation or perturbation of data, depending on the distance to x. $\Omega(g)$ is a penalty given to the model complexity of g, for example it is often preferred to have fewer features in an explanation model, so Ω would be lower in this case.

In particular, if a linear regression model is chosen as the explanation function g, then the LIME model would be a form of:

$$f(x) \approx g(z') = b_0 + \sum_{i=1}^{M} b_i z'_i.$$

In this formula z' is a vector of indicators in $\{0, 1\}^M$, which indicates if a feature is used in the explanation model, and M is the total number of features. b_i is the weight given to each feature included in the explanation model.

The biggest question in the LIME model, is how to weight the different variations or perturbations of the data. For example, it seems logical to give a high weight to the variations that are close to the fixed instance to be explained x.

4.2.2. Kernel SHAP

Kernel SHAP [9] gives an answer to the last question from LIME. Namely, Kernel SHAP proposes to use weighted linear regression as an explanation function. The weights associated with this weighted linear regression were still unclear in the LIME model from the previous section, but Kernel SHAP proposes to give the Shapley weight distribution to the different variations or perturbations of the data. The theory behind this is that solving this specific LIME model gives us a new, and quicker way to calculate Shapley values, as linear regression is applied to find Shapley values. The model then becomes:

$$g(z') = \phi_0 + \sum_{i=1}^{N} \phi_i z'_i,$$

where ϕ_i is the Shapley value of feature i. In order to find these ϕ's we apply the following method:

1. First, sample random $z'_k \in \{0,1\}^N$ with $k \in \{1, ..., K\}$. Thus sampling K random coalitions of features, where a 1 indicates the feature is used and for 0 it is not. This is random in such a way that each feature has a 50% chance of being included.
2. Then convert the z'_k to the original feature space by taking $h_x(z'_k)$. This is done by mapping the 1's to the corresponding feature value in x and mapping the 0's to a random value from the marginal distribution.
3. Now apply the machine learning model $f(h_x(z'_k))$.
4. Lastly, compute the weight of z'_k by using the SHAP kernel. This results in the following weight based on the number of 1's in z'_k, also the L^1-norm of z'_k:

$$\pi(z') = \frac{(N-1)}{\binom{N}{|z'|}|z'|(N-|z'|)}$$

After following this algorithm n times, a dataset can be created with the different perturbations of data, their predictions and the corresponding weights. Then it is possible to apply the linear regression model to that dataset by minimizing the following loss:

$$L(f, g, \pi) = \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi(z')$$

where $g(z') = \phi_0 + \sum_{i=1}^{N} \phi_i z'_i$.

4.2.3. Shapley Weights

As mentioned before Kernel SHAP uses a weighted linear regression, with the following weights representing the Shapley weights:

$$\pi(z') = \frac{(N-1)}{\binom{N}{|z'|}|z'|(N-|z'|)}.$$

However, with these weights taking $|z'| = 0$ or $|z'| = M$, results in this weight being infinite. Since infinite weights can not be applied in machine learning, these weights are applied by adding two extra constraints to the model:

1. The first constraint describes what happens when $|z'| = N$ and states that the local accuracy still holds:

$$f(x) = \sum_{i=1}^{n} \phi_i.$$

2. The second constraint describes what happens when $|z'| = 0$ and states that:

$$\phi_0 = \mathbb{E}[f(X)]$$

Instead of adding these constraints,in a machine learning setting it is also possible to, in case of infinite weights, make the weights arbitrarily high. Ensuring that by default these constraints are implied.

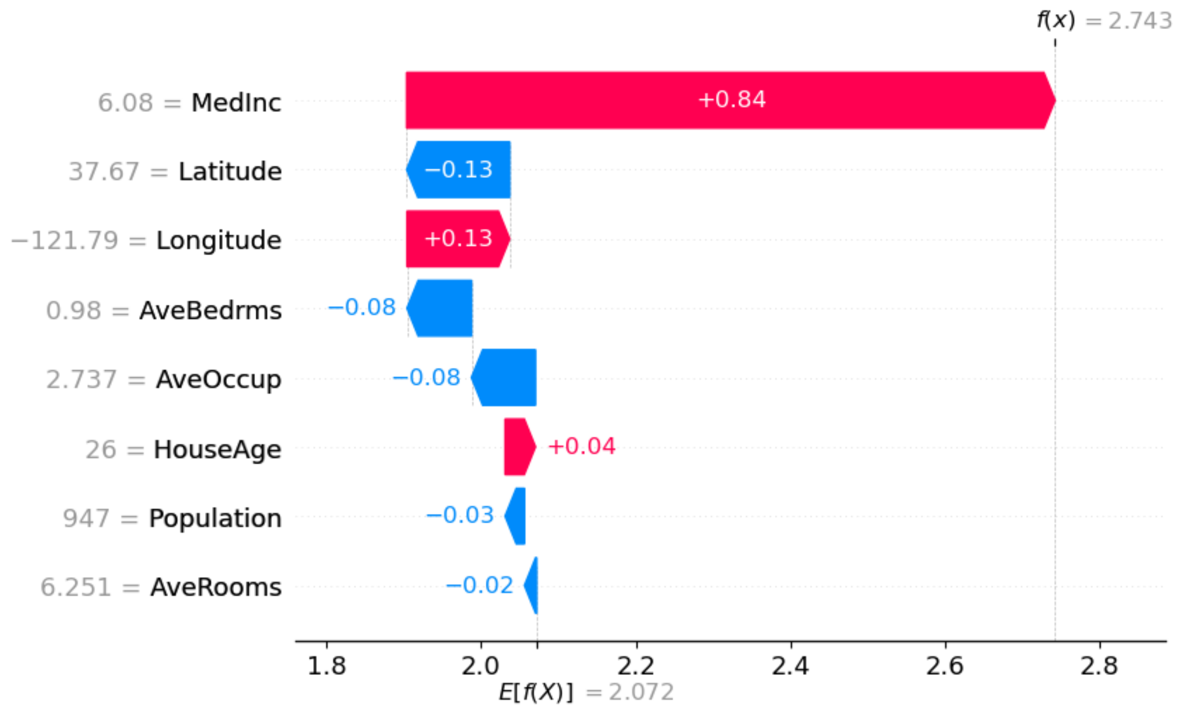


Figure 4.2: Waterfall plot of Shapley values using Kernel SHAP Method.

4.2.4. Implementation in Python

Applying the Kernel SHAP algorithm in Python yields a waterfall plot that can be seen in Figure 4.2. This figure again uses the same fixed instance x as in Figure 3.1 and Figure 4.1. Again, while the values are not completely the same, the features that are the most important are consistent with earlier results.

4.2.5. Regularization

Since Kernel SHAP is in a linear regression setting, it is possible to use different adaptations that can also be applied to linear regression, such as regularization. In this case, it is possible to regularize the least squares solution through Lasso:

$$L(\hat{f}, g, \pi) = \sum_{z' \in Z} [\hat{f}(h_x(z')) - g(z')]^2 \pi(z') + \lambda \sum_{i=1}^{M} |\phi_i|$$

This signifies that a penalty parameter λ is added to the loss function for all coefficients. Therefore given that the penalty term (λ) is high enough, coefficients can become 0, and the explanations are more interpretable. When a model consists of a lot of features this is especially useful, since the least important features are eliminated, making the model more interpretable. The parameter λ controls the strength of the regularizing effect, and can be tuned by cross-validation.

The Kernel SHAP method in Python also has built-in Lasso regularization, thus an explanation can also be made using only four features as shown in Figure 4.3. The biggest difference between this regularized explanation and the explanation in Figure 4.2, is that this explanation only consists of four features, instead of 8. The values of the 4 most important features are almost the same as in Figure 4.2.

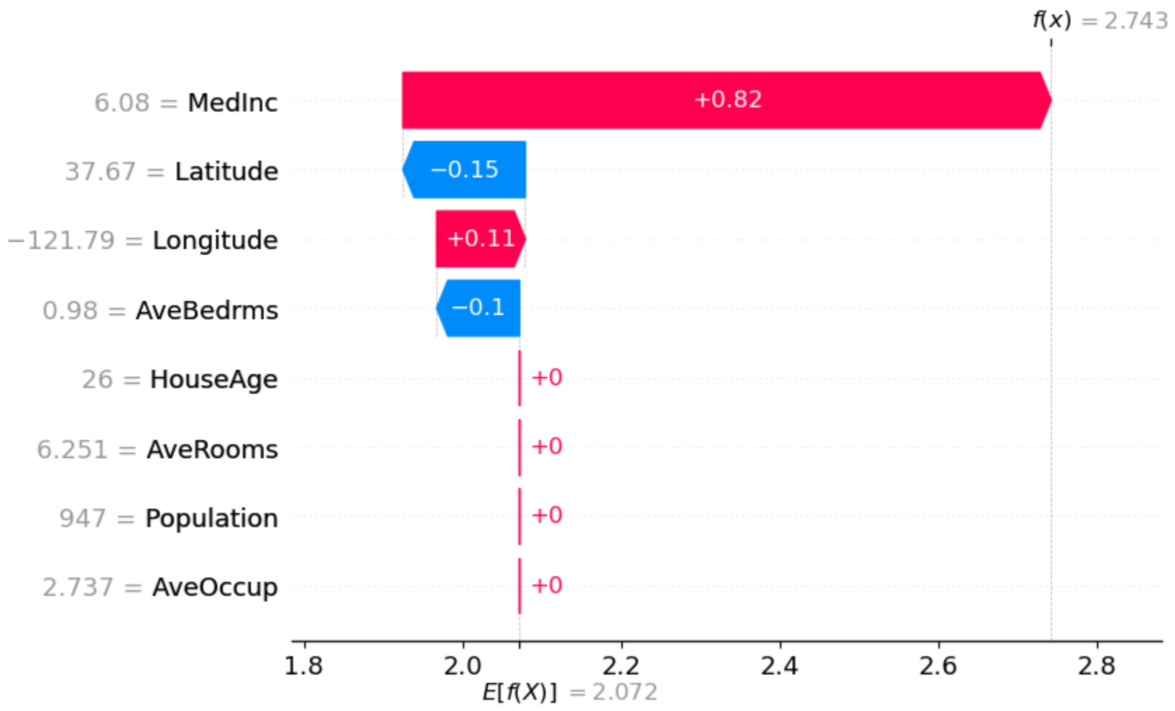
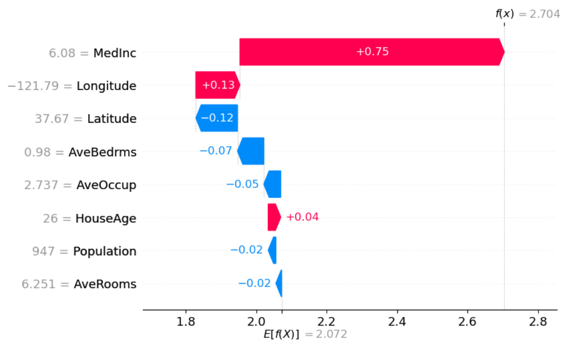


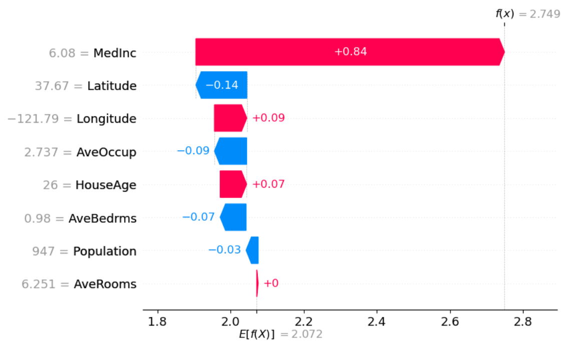
Figure 4.3: Waterfall plot of Shapley values using Kernel SHAP Method with Regularization

5 Implementation

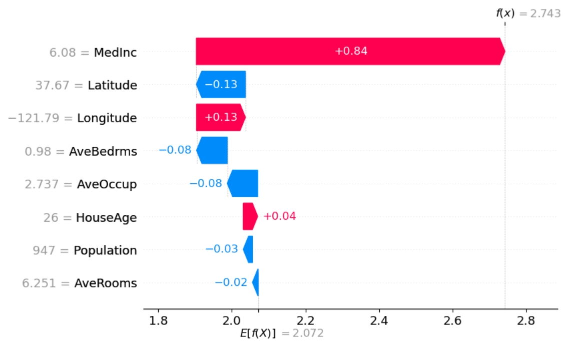
In previous chapters three different methods have been introduced to approximate Shapley values: two different Monte Carlo methods and Kernel SHAP. A summary of applying these three methods to the same instance can be seen in Figure 5.1. At first glance, these three waterfall plots look quite similar. All of them place the highest importance on the features MedInc, Longitude and Latitude.



(a) Monte Carlo Value Distribution



(b) Monte Carlo through Permutation



(c) Kernel SHAP

Figure 5.1: Results of three different ways to calculate Shapley values for the same instance

However, there are still differences between the three figures, for example the exact Shapley value of MedInc, or the order of the smaller Shapley values. When working with Monte Carlo methods, a way to increase the accuracy could be to increase the number of iterations. However, while the results do become slightly more stable, since the Monte Carlo methods are less efficient, the computation times also rise drastically.

Furthermore, even though the Monte Carlo results become more stable they do not necessarily converge exactly to the Kernel SHAP calculation. Still, while the values may not agree exactly, in broad terms the explanations are still very similar. Also, since all three are valid explanations, it is difficult to check which explanation is the best. Nevertheless, one would assume that with a very high number of iterations, the explanation would resemble the true Shapley values the closest.

Monte Carlo Value Distribution	1165 seconds
Monte Carlo through Permutations	1045 seconds
Kernel SHAP	18.31 seconds

Table 5.1: Computation times of the three methods

In table 5.1 the computation times are given that were needed to produce the figures in Figure 5.1. This table shows that Kernel SHAP is by far the quickest computation method (more than 50 times quicker). Furthermore, there is not a big difference between the Monte Carlo methods, but this could be due to the fact that the California dataset only has 8 features. Therefore, the amount of coalitions is still feasible, and thus the amount of iterations for both Monte Carlo Methods can be similar. However, since this is quite a small dataset with a limited amount of features, it is also interesting to see the performance of these methods on a bigger dataset with more features. The main goal of applying the Shapley techniques to a bigger dataset is to see if the three methods are still feasible, and again produce similar results.

5.1. Ames Housing Data

To see how the Shapley values behave with more features, a different model is used: the Ames housing dataset. This dataset consists of detailed information about residential properties in Ames, Iowa. The dataset is a widely-used dataset in data science and machine learning, mainly because of its many features. It includes over 80 attributes that describe various aspects of the houses such as size, number of rooms, neighborhood, year built, and other physical attributes. The target variable corresponding to these features is the Sale Price.

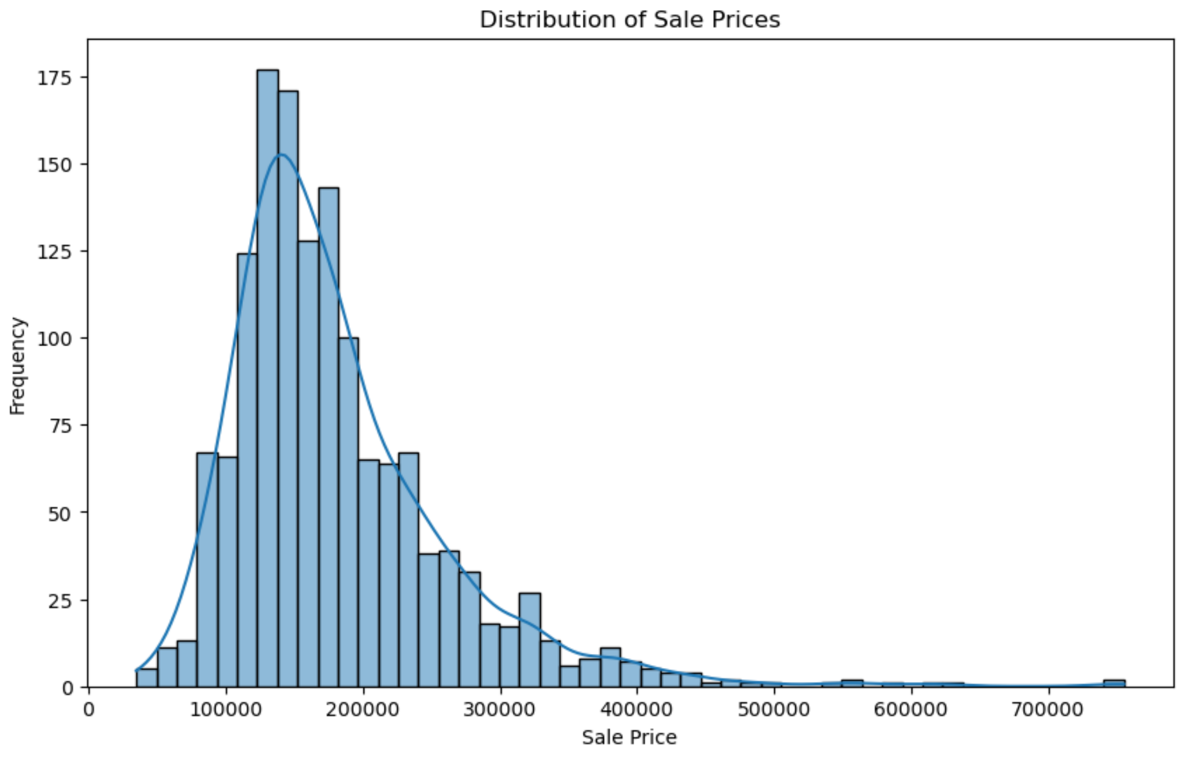


Figure 5.2: Histogram of target variable Sale Price

In order to get a better feeling for the data some visualizations were made. In Figure 5.2 a histogram is shown of the target variable Sale Price. This histogram has a heavy right tail, showing that there are many high outliers in this dataset. Furthermore, Figure 5.3 shows the relationship between Sale Price and a single feature: the living area above ground (GrLivArea). This scatterplot confirms the findings of the histogram, with two very high outliers, and most of the data concentrated on the lower Sale Price. Interesting to note is that the more the above ground living area increases, the farther the data points are spread out.

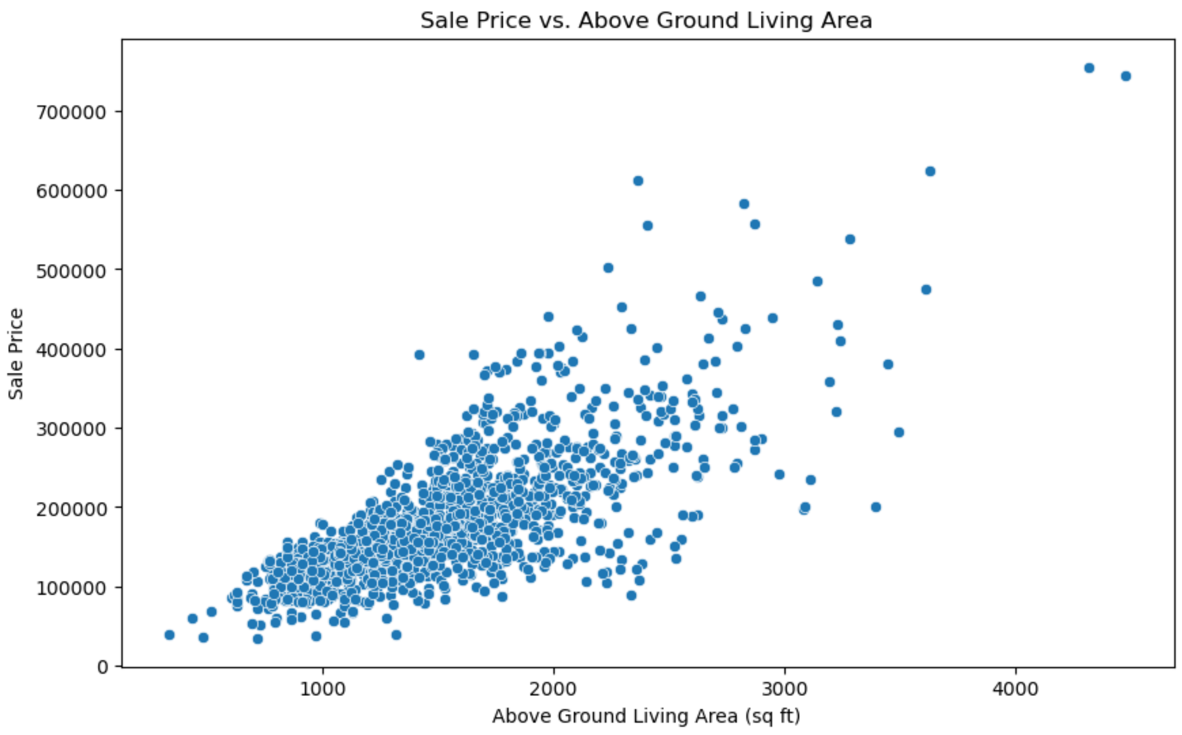


Figure 5.3: Scatterplot of relation between Sale Price and Above Ground Living Area

One of the reasons why the random forest is so efficient for this data is because they are well-suited for high dimensional data. Random forests do not require extensive preprocessing or feature selection to capture complex interactions among many features. Random forests also have an outlier robustness, meaning they maintain performance, even in a dataset with outliers. Furthermore, they are able to handle non-linear relationships. Since this data contains outliers, has many features and seems to behave non-linearly again, a random forest is chosen to model this data.

As this dataset consists of real data a cleaned/preprocessed dataset will be used in this thesis. The process of cleaning a dataset can consist of removing/replacing missing values, removing duplicates, removing irrelevant features and encoding categorical variables. Since this dataset also has a lot of categorical variables, many dummy variables need to be created. Thus after the cleaning process, the dataset consists of 302 variables.

5.1.1. Monte Carlo Value Function

The first problem with calculating SHAP values for this dataset arises when trying to use the Monte Carlo value function method. For this method, all combinations need to be considered. However, for the Ames housing dataset, 302 features means 2^{302} possible coalitions. It would take too long for a computer to be able to calculate the value function for all these coalitions, and thus this method is not feasible for this amount of features.

5.1.2. Monte Carlo through Permutations

Exactly for this reason, a second Monte Carlo method was developed. This method can still function despite the many features, since only the number of iterations is required, and it does not have to make these iterations for every coalition of features.

However, since this dataset contains more features, our machine learning function f is more complicated, and thus the model takes longer to execute. Since the main part of the Monte Carlo method consists of applying the machine learning function to different subsets or permutations of data, the computation times will also rise. In practice this means that if 1000 iterations are taken, calculating the Monte Carlo Shapley values takes around 5069 seconds. Effectively, this is around 50 times slower than for the previous dataset, since for the California dataset 10 000 iterations were taken.

One could argue that since there are more features, and thus more possible permutations, it would be necessary to have more iterations. In theory, more iterations would then be able to better capture relationships between features and produce more accurate Shapley values. Even so, 1000 iterations already takes more than an hour to compute, so the increase in accuracy of the Shapley values would be insignificant compared to the increase in computation time. Thus the number of iterations was chosen to remain 1000.

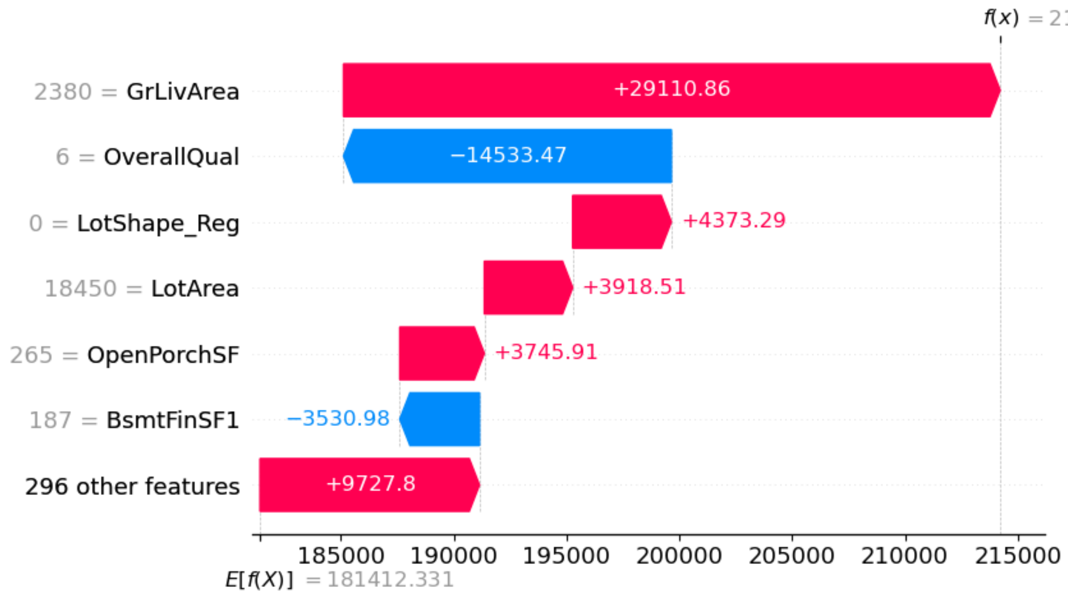


Figure 5.4: Waterfall plot of Shapley values for Ames housing data through Monte Carlo calculation

Taking a specific instance x and calculating the Shapley values through Monte Carlo yields Figure 5.4. The average predicted sale price for this dataset is 181412, and this specific house is predicted to be 214224. Since there are 302 features that contribute in this prediction, the top six features are chosen to be presented in this plot, and the other features are grouped together. The two main features contributing to this prediction are GrLivArea (above ground living area), and OverallQual (quality score of the house based on a score from 1 to 10).

The above ground living area of 2360 contributes the most positively. Checking with the scatterplot in Figure 5.3 this makes sense, since looking at the houses with a sale price of 214,000 this living area does seem quite big. Additionally, the the quality of the house (6) contributes the most negatively. This means that this is quite a low score for this house, and thus affects the sale price negatively.

Then, there are three more features that impact this house positively, namely: LotShape_Reg (this is a dummy variable and is 0, meaning the house has an irregular shape), LotArea (18450 square feet), and the porch area (265 square feet). The final feature in this waterfall plot that impacts the sale price of this house negatively is the square feet of the basement, namely 187 square feet.

5.1.3. Kernel SHAP

Shapley values for the Ames housing dataset can also be calculated through the Kernel SHAP method. Calculating the Shapley values for a single instance using this method takes around 50 seconds. Figure 5.5 shows the waterfall plot of the Shapley values applied to the same instance as in the previous section.

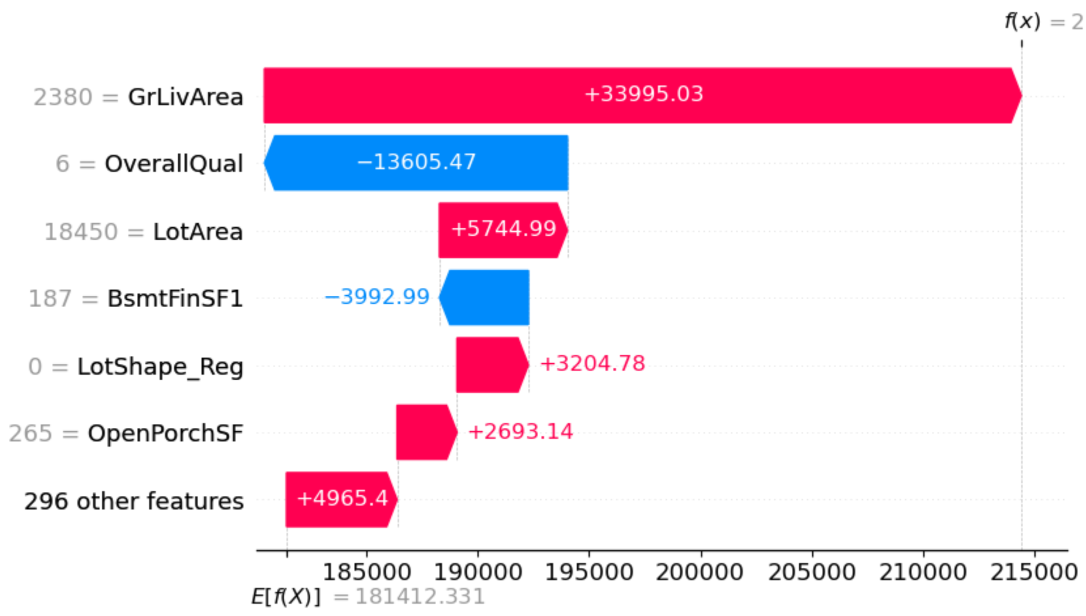


Figure 5.5: Waterfall plot of Shapley values for Ames housing data through Kernel SHAP

While there are some small differences, this figure exhibits mostly similarities with the previous finding. For instance, the same six features are found to be the most important. Small differences include the order of the most important features, and the Shapley values are not exactly the same, however in broad terms the values do tend to agree.

5.2. Comparisons

The main goal of applying the Shapley techniques to the Ames housing dataset was to see if the three methods are still feasible, and again produce similar results. While the amount of features was too much for the first Monte Carlo method to handle, for the other two methods, the results were indeed similar, as was also the case with the smaller dataset.

However, a significant difference lies in the computational efficiency of the two methods. The Kernel SHAP method was markedly faster than the Monte Carlo method for both datasets. This difference in speed becomes particularly important when dealing with datasets with more features and more instances. After evaluating the results, the two datasets can be compared.

California Housing Dataset: This dataset has fewer features compared to the Ames Housing dataset. Both Monte Carlo and Kernel SHAP methods perform efficiently, but Kernel SHAP was still a lot quicker, making it a preferable choice even for datasets with a low number of features.

Ames Housing Dataset: The Ames Housing dataset contains a larger number of features, which increases the computational benefits of Kernel SHAP. The Monte Carlo method's performance significantly slowed as the number of features increased, while Kernel SHAP maintained its efficiency, completing the calculations much faster.

In summary, while both methods provide accurate Shapley values, Kernel SHAP's superior speed, especially with larger datasets like Ames Housing, makes it a more practical choice for real-world applications where computational resources and time are critical considerations.

6 Conclusion and discussion

In this thesis, the application of Shapley values as a method for interpreting machine learning models was explored, delving into their mathematical foundation, computation methods, and practical implementation. When machine learning models are applied, an explanation can be extracted via multiple explanatory methods. The Shapley values method seems to be the most suitable method because this method has a strong theoretic base with desirable properties.

Based on this theory three methods were introduced to calculate Shapley values, two Monte Carlo methods and Kernel SHAP. These methods were then applied to two different datasets: California housing dataset and Ames housing dataset. Analyzing confirmed that both methods yield similar results, validating the correctness of the Shapley values. However, Kernel SHAP was significantly faster, highlighting its practical advantage in real-world applications where computational efficiency is important.

While Shapley values have a strong theoretical base and constitute a comprehensive approach to model interpretability, several limitations have to be acknowledged.

One of these limitations is the computational complexity of the Shapley values. The exact computation of Shapley values is often infeasible for models with many features due to the increase in the number of permutations. Approximate methods like Monte Carlo and Kernel SHAP can work around this, but can also still be hard to compute for large datasets and complex models.

Another limitation is that feature dependence is not considered when calculating Shapley values. By replacing feature values with values from random instances, the marginal distribution is sampled. However, if features are correlated, this leads to putting too much weight on unlikely instances.

An additional limitation is that Shapley values do not provide insight in the inner workings of a black box model. They explain the output in terms of feature contributions but do not explain the internal structure of the model, keeping its black-box nature.

Future research can address these limitations by exploring different approaches to overcome these shortcomings. Some options are: research into more efficient algorithms for Shapley value approximation, methods for handling feature correlations, and techniques to enhance the insights derived from Shapley values. These options can all improve the usefulness of Shapley values in practical applications.

In conclusion, Shapley values are a powerful tool for interpreting machine learning models. They have a strong theoretical foundation and are easy to understand. The analysis comparing Monte Carlo and Kernel SHAP methods shows that Shapley values can be integrated into machine learning processes, with Kernel SHAP being much faster. For future research, by handling the limitations of Shapley values, it is possible to make machine learning models more transparent and trustworthy.

Bibliography

[1] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[2] Leo Breiman. “Random Forests”. In: *Machine Learning* (2001), pp. 5–32.

[3] R.R. Picard and K.N. Berk. “Splitting Data”. In: *Am. Statis* (1990), pp. 140–147.

[4] G. Afendras and M. Markatou. “Optimality of training/test size and resampling effectiveness in cross-validation”. In: *J. Statis. Plan. Infer.* (2019), pp. 286–301.

[5] Christophe Molnar. *Interpretable Machine Learning*. 2019. URL: `https://christophm.github.io/interpretable-ml-book/`.

[6] Lloyd Shapley. “A Value for N-Person Games”. In: *Princeton University Press* (1953), pp. 307–317.

[7] Erik Štrumbelj and Igor Kononenko. “Explaining prediction models and individual predictions with feature contributions”. In: *Knowledge and information systems* (2014), pp. 647–665.

[8] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics, June 2016, pp. 97–101.

[9] Scott Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems* (2017).

A

Summation of Shapley weights

Proof that the sum of Shapley weights is 1:

$$
\begin{aligned}
\sum_{S\subseteq N\setminus\{i\}} \frac{|S|!(N-|S|-1)!}{N!} &= \sum_{k=1}^{n} \sum_{S\subseteq N\setminus\{i\}} \frac{k!(N-k-1)!}{N!} \\
&= \sum_{k=1}^{n} \frac{k!(N-k-1)!}{N!} \cdot \binom{N-1}{k} \\
&= \sum_{k=1}^{n} \frac{k!(N-k-1)!}{N!} \cdot \frac{(N-1)!}{k!(N-k-1)!} \\
&= \sum_{k=1}^{n} \frac{1}{N} = 1.
\end{aligned}
$$

B
Code

For the implementation of the random forest to the dataset the following code was used:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

X, y = shap.datasets.california(n_points=2000)

X = pd.DataFrame(X,columns = ["MedInc", "HouseAge", "AveRooms", "AveBedrms", "Population", "AveOccup", "Lat","Long"])

# Random forest model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = sklearn.ensemble.RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

For the Monte Carlo value function method the following code was used:

```python
from itertools import combinations
from sklearn.base import is_regressor, is_classifier
from math import comb
import time

def compute_shapley_values(model, x, X, num_samples_per_subset=10):
    """
    Compute the Shapley values for all features of the instance x using exact Shapley weights
    and multiple random instances per subset.

    Parameters:
    model : a trained model (must have a predict or predict_proba method)
    x : numpy array, instance of interest
    X : pandas DataFrame, data matrix
    num_samples_per_subset : int, number of random instances per subset

    Returns:
    shapley_values : numpy array, the Shapley values for all features of x
    """
    def predict_fn(x):
        if is_classifier(model):
            return model.predict_proba(x)[:, 1]
        elif is_regressor(model):
            return model.predict(x)
        else:
            raise ValueError("Model must be a regressor or classifier.")

    n_features = X.shape[1]
    x = np.array(x).reshape(1, -1)  # Ensure x is a 2D array
    shapley_values = np.zeros(n_features)

    for j in range(n_features):
        marginal_contributions = []

        # Iterate over all subsets S of features excluding j
        for k in range(n_features):
            if k == j:
                continue

            for S in combinations([i for i in range(n_features) if i != j], k):
                S = list(S)

                with_j_preds = []
                without_j_preds = []

                for _ in range(num_samples_per_subset):
                    # Draw a random instance z from the data matrix X
                    z = X.sample(n=1).values.flatten()

                    # Construct two new instances
                    x_plus_j = x.copy()
                    x_minus_j = x.copy()

                    # Features in S remain from x, other features from z
                    for i in range(n_features):
                        if i in S or i == j:
                            x_plus_j[0, i] = x[0, i]
                        else:
                            x_plus_j[0, i] = z[i]
                        if i in S:
                            x_minus_j[0, i] = x[0, i]
                        else:
                            x_minus_j[0, i] = z[i]

                    # Compute predictions
                    with_j_pred = predict_fn(pd.DataFrame(x_plus_j, columns=X.columns))
                    without_j_pred = predict_fn(pd.DataFrame(x_minus_j, columns=X.columns))
                with_j_preds.append(with_j_pred[0])
                without_j_preds.append(without_j_pred[0])

            # Average the predictions
            avg_with_j_pred = np.mean(with_j_preds)
            avg_without_j_pred = np.mean(without_j_preds)

            # Compute the marginal contribution
            phi_j_S = avg_with_j_pred - avg_without_j_pred

            # Compute the weight
            weight = comb(n_features - 1, len(S)) ** -1 / n_features

            # Add the weighted marginal contribution
            marginal_contributions.append(weight * phi_j_S)

        # Compute the Shapley value as the sum of the weighted marginal contributions
        shapley_values[j] = np.sum(marginal_contributions)
        print(shapley_values[j])

    return shapley_values



# Instance of interest
x_instance = X_test.iloc[0].values

start_time = time.time()

# Calculate the Shapley values for all features
shapley_values = compute_shapley_values(model, x_instance, X_train, num_samples_per_subset=100)
shapley_values_df = pd.DataFrame(shapley_values, index=X_train.columns, columns=["Shapley Value"])
end_time = time.time()


shap_value_mc1 = shap.Explanation(
    values=shapley_values,
    base_values=model.predict(X_train).mean(),
    data=x_instance,
    feature_names=X_train.columns.tolist()
)

elapsed_time = end_time - start_time

print(f"Elapsed time: {elapsed_time} seconds")

# Plot the waterfall plot
shap.waterfall_plot(shap_value_mc1)
```

For the Monte Carlo method through permutations the following code was used:

```
def compute_shapley_value(model, x, j, X, M=100):
    """
    Compute the Shapley value for the j-th feature of the instance x.

    Parameters:
    model : a trained model (must have a predict or predict_proba method)
    x : numpy array, instance of interest
    j : int, feature index
    X : pandas DataFrame, data matrix
    M : int, number of iterations

    Returns:
    shapley_value : float, the Shapley value for the j-th feature
    """
    def predict_fn(x):
        if is_classifier(model):
            return model.predict_proba(x)[:, 1]
        elif is_regressor(model):
            return model.predict(x)
        else:
            raise ValueError("Model must be a regressor or classifier.")

    n_features = X.shape[1]
    x = np.array(x).reshape(1, -1)  # Ensure x is a 2D array
    shapley_values = []

    for _ in range(M):
        # Draw a random instance z from the data matrix X
        z = X.sample(n=1).values.flatten()

        # Choose a random permutation o of the feature indices
        o = np.random.permutation(n_features)
        # Order instance x and z according to the permutation o
        x_o = x[:, o]
        z_o = z[o]

        # Construct two new instances
        x_plus_j = x_o.copy()
        x_minus_j = x_o.copy()

        # Features up to j remain from x, features after j from z
        x_plus_j[0, np.where(o > j)[0]] = z_o[np.where(o > j)[0]]
        x_minus_j[0, np.where(o >= j)[0]] = z_o[np.where(o >= j)[0]]

        # Restore the original order
        x_plus_j = x_plus_j[:, np.argsort(o)]
        x_minus_j = x_minus_j[:, np.argsort(o)]

        # Compute predictions
        with_j_pred = predict_fn(pd.DataFrame(x_plus_j, columns=X.columns))
        without_j_pred = predict_fn(pd.DataFrame(x_minus_j, columns=X.columns))

        # Compute the marginal contribution
        phi_j_m = with_j_pred - without_j_pred
        shapley_values.append(phi_j_m[0])

    # Compute the Shapley value as the average of the marginal contributions
    shapley_value = np.mean(shapley_values)

    return shapley_value


# Instance of interest
x_instance = X_test.iloc[0].values
```

```python
# Calculate the Shapley value

start_time = time.time()

n_features = X_train.shape[1]
shapley_values_mc = np.zeros(n_features)
for j in range(n_features):
    shapley_values_mc[j] = compute_shapley_value(model, x_instance, j, X_train, M=10000)
    print(shapley_values_mc[j])

end_time = time.time()

shap_value_mc_2 = shap.Explanation(
    values=shapley_values_mc,
    base_values=model.predict(X_train).mean(),
    data=x_instance,
    feature_names=X.columns.tolist()
)

elapsed_time = end_time - start_time

print(f"Elapsed time: {elapsed_time} seconds")

# Plot the waterfall plot
shap.waterfall_plot(shap_value_mc_2)
```

For the Kernel SHAP method the following code was used:

```
import shap
explainer2 = shap.KernelExplainer(model.predict,X_train)

start_time = time.time()

x_instance = X_test.iloc[0]
shap_values2 = explainer2.shap_values(x_instance,l1_reg='num_features(4)')

end_time = time.time()

shap_values_2 = shap.Explanation(values=shap_values2, base_values=model.predict(X_train).mean(), data=x_instance)

elapsed_time = end_time - start_time

print(f"Elapsed time: {elapsed_time} seconds")

shap.plots.waterfall(shap_values_2)
```