# 5.10 SHAP (SHapley Additive exPlanations)

*This chapter is currently only available in this web version. ebook and print will follow.*

SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2016)[48] is a method to explain individual predictions. SHAP is based on the game theoretically optimal Shapley Values.

There are two reasons why SHAP got its own chapter and is not a subchapter of Shapley values. First, the SHAP authors proposed KernelSHAP, an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models. And they proposed TreeSHAP, an efficient estimation approach for tree-based models. Second, SHAP comes with many global interpretation methods based on aggregations of Shapley values. This chapter explains both the new estimation approaches and the global interpretation methods.

> Interested in an in-depth, hands-on course on SHAP and Shapley values? Head over to the Shapley course page and get notified once the course is available.

I recommend reading the chapters on Shapley values and local models (LIME) first.

## 5.10.1 Definition

The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction. The SHAP explanation method computes Shapley values from coalitional game theory. The feature values of a data instance act as players in a coalition. Shapley values tell us how to fairly distribute the "payout" (= the prediction) among the features. A player can be an individual feature value, e.g. for tabular data. A player can also be a group of feature values. For example to explain an image, pixels can be grouped to super pixels and the prediction distributed among them. One innovation that SHAP brings to the table is that the Shapley value explanation is represented as an additive feature attribution method, a linear model. That view connects LIME and Shapley Values. SHAP specifies the explanation as:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

where g is the explanation model, $z' \in \{0,1\}^M$ is the coalition vector, M is the maximum coalition size and $\phi_j \in \mathbb{R}$ is the feature attribution for a feature j, the Shapley values. What I call "coalition vector" is called "simplified features" in the SHAP paper. I think this name was chosen, because for e.g. image data, the images are not represented on the pixel level, but aggregated to super pixels. I believe it is helpful to think about the z's as describing coalitions: In the coalition vector, an entry of 1 means that the corresponding feature value is "present" and 0 that it is "absent". This should sound familiar to you if you know about Shapley values. To compute Shapley values, we simulate that only some features values are playing ("present") and some are not ("absent"). The representation as a linear model of coalitions is a trick for the computation of the $\phi$'s. For x, the instance of interest, the coalition vector x' is a vector of all 1's, i.e., all feature values are "present". The formula simplifies to:

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j$$

You can find this formula in similar notation in the Shapley value chapter. More about the actual estimation comes later. Let us first talk about the properties of the $\phi$'s before we go into the details of their estimation.

Shapley values are the only solution that satisfies properties of Efficiency, Symmetry, Dummy and Additivity. SHAP also satisfies these, since it computes Shapley values. In the SHAP paper, you will find discrepancies between SHAP properties and Shapley properties. SHAP describes the following three desirable properties:

**1) Local accuracy**

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j$$

If you define $\phi_0 = E_X(\hat{f}(x))$ and set all $x'_j$ to 1, this is the Shapley efficiency property. Only with a different name and using the coalition vector.

$$f(x) = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j = E_X(\hat{f}(X)) + \sum_{j=1}^{M} \phi_j$$

**2) Missingness**

$$x'_j = 0 \Rightarrow \phi_j = 0$$

Missingness says that a missing feature gets an attribution of zero. Note that $x'_j$ refers to the coalitions, where a value of 0 represents the absence of a feature value. In coalition notation, all feature values $x'_j$ of the instance to be explained should be '1'. The presence of a 0 would mean that the feature value is missing for the instance of interest. This property is not among the properties of the "normal" Shapley values. So why do we need it for SHAP? Lundberg calls it a "minor book-keeping property". A missing feature could -- in theory -- have an arbitrary Shapley value without hurting the local accuracy property, since it is multiplied with $x'_j=0$. The Missingness property enforces that missing features get a Shapley value of 0. In practice this is only relevant for features that are constant.

**3) Consistency**

Let $f_x(z')=f(h_x(z'))$ and $z'_{\setminus j}$ indicate that $z'_j=0$. For any two models f and f' that satisfy:

$$f'_x(z')-f'_x(z'_{\setminus j})\geq f_x(z')-f_x(z'_{\setminus j})$$

for all inputs $z'\in\{0,1\}^M$, then:

$$\phi_j(f',x)\geq\phi_j(f,x)$$

The consistency property says that if a model changes so that the marginal contribution of a feature value increases or stays the same (regardless of other features), the Shapley value also increases or stays the same. From Consistency the Shapley properties Linearity, Dummy and Symmetry follow, as described in the Appendix of Lundberg and Lee.

## 5.10.2 KernelSHAP

KernelSHAP estimates for an instance x the contributions of each feature value to the prediction. KernelSHAP consists of 5 steps:

- Sample coalitions $z'_k\in\{0,1\}^M,\quad k\in\{1,\ldots,K\}$ (1 = feature present in coalition, 0 = feature absent).
- Get prediction for each $z'_k$ by first converting $z'_k$ to the original feature space and then applying model f: $f(h_x(z'_k))$
- Compute the weight for each $z'_k$ with the SHAP kernel.
- Fit weighted linear model.
- Return Shapley values $\phi_k$, the coefficients from the linear model.

We can create a random coalition by repeated coin flips until we have a chain of 0's and 1's. For example, the vector of (1,0,1,0) means that we have a coalition of the first and third features. The K sampled coalitions become the dataset for the regression model. The target for the regression model is the prediction for a coalition. ("Hold on!," you say, "The model has not been trained on these binary coalition data and can't make predictions for them.") To get from coalitions of feature values to valid data instances, we need a function $h_x(z') = z$ where $h_x : \{0,1\}^M \rightarrow \mathbb{R}^p$. The function $h_x$ maps 1's to the corresponding value from the instance x that we want to explain. For tabular data, it maps 0's to the values of another instance that we sample from the data. This means that we equate "feature value is absent" with "feature value is replaced by random feature value from data". For tabular data, the following figure visualizes the mapping from coalitions to feature values:

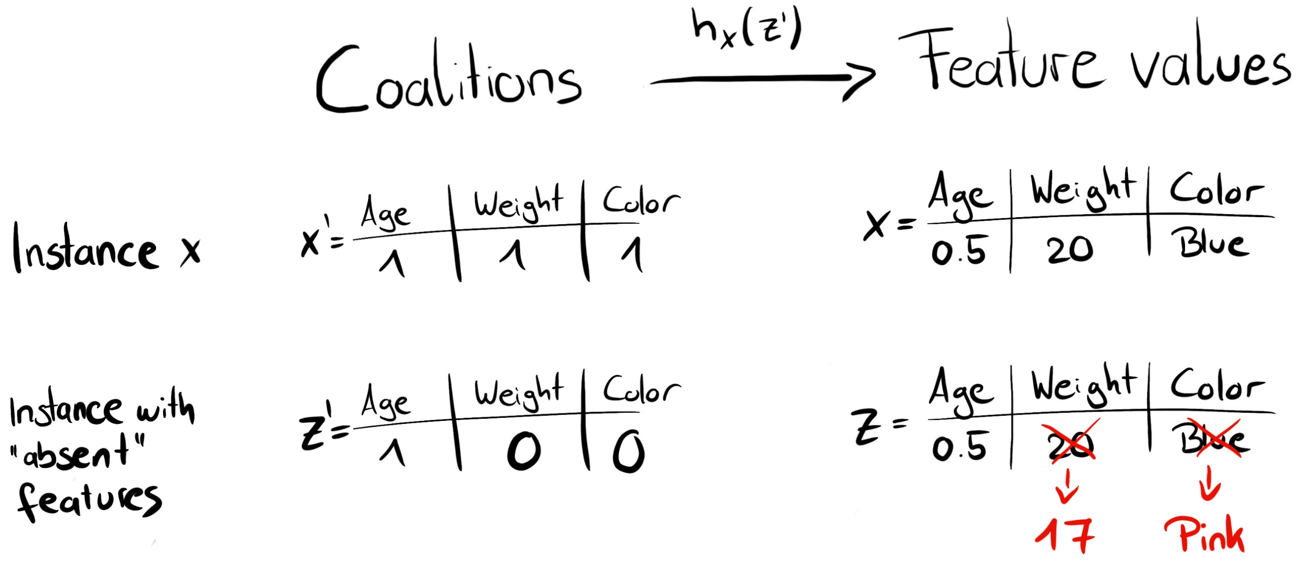

Coalitions
$h_x(z')$
Feature values
Instance x
x' = Age 1 | Weight 1 | Color 1
x = Age 0.5 | Weight 20 | Color Blue
Instance with "absent" features
z' = Age 1 | Weight 0 | Color 0
z = Age 0.5 | Weight ~~20~~ | Color ~~Blue~~
17
Pink

FIGURE 5.48: Function $h_x$ maps a coalition to a valid instance. For present features (1), $h_x$ maps to the feature values of x. For absent features (0), $h_x$ maps to the values of a randomly sampled data instance.

$h_x$ for tabular data treats $X_C$ and $X_S$ as independent and integrates over the marginal distribution:

$$f(h_x(z')) = E_{X_C}[f(x)]$$

Sampling from the marginal distribution means ignoring the dependence structure between present and absent features. KernelSHAP therefore suffers from the same problem as all permutation-based interpretation methods. The estimation puts too much weight on unlikely instances. Results can become unreliable. But it is necessary to sample from the marginal distribution. If the absent feature values would be sampled from the conditional distribution, then the resulting values are no longer Shapley values. The resulting values would violate the Shapley axiom of Dummy, which says that a feature that does not contribute to the outcome should have a Shapley value of zero.

For images, the following figure describes a possible mapping function:

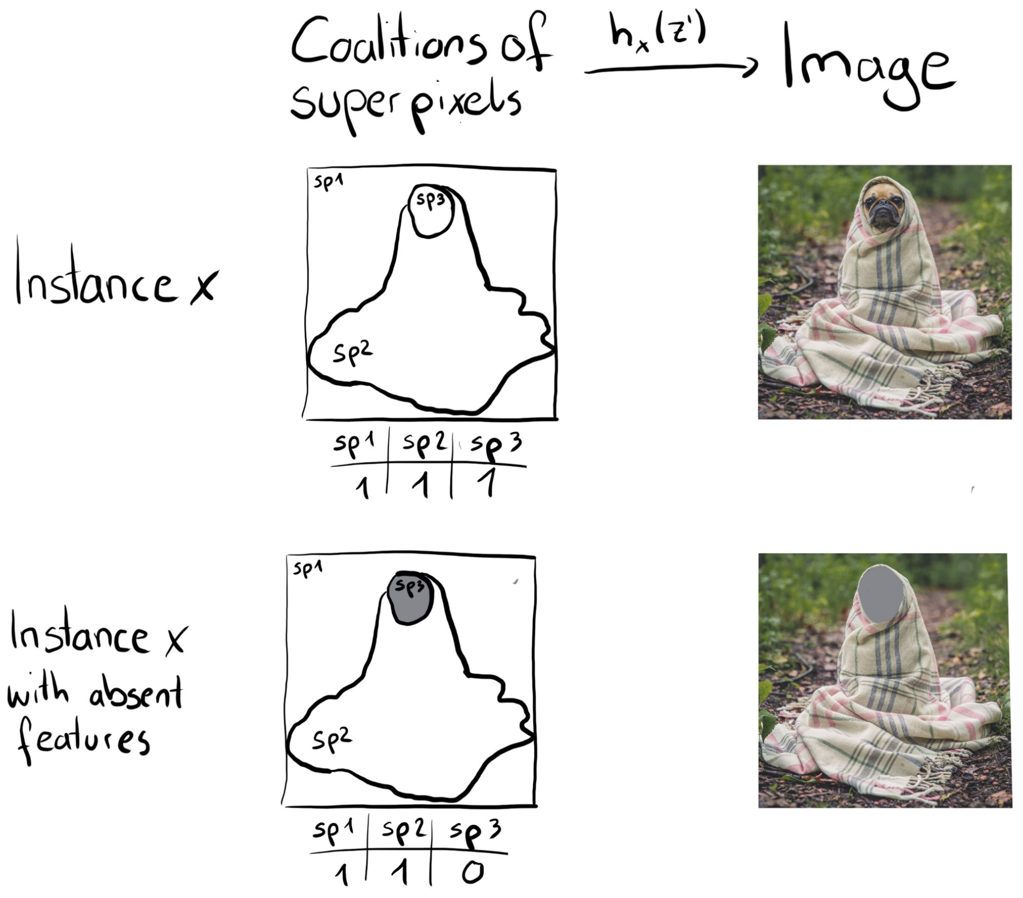


FIGURE 5.49: Function $h_x$ maps coalitions of super pixels (sp) to images. Super-pixels are groups of pixels. For present features (1), $h_x$ returns the corresponding part of the original image. For absent features (0), $h_x$ greys out the corresponding area. Assigning the average color of surrounding pixels or similar would also be an option.

The big difference to LIME is the weighting of the instances in the regression model. LIME weights the instances according to how close they are to the original instance. The more 0's in the coalition vector, the smaller the weight in LIME. SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights. The intuition behind it is: We learn most about individual features if we can study their effects in isolation. If a coalition consists of a single feature, we can learn about the features' isolated main effect on the prediction. If a coalition consists of all but one feature, we can learn about this features' total effect (main effect plus feature interactions). If a coalition consists of half the features, we learn little about an individual features contribution, as there are many possible coalitions with half of the features. To achieve Shapley compliant weighting, Lundberg et. al propose the SHAP kernel:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)}$$

Here, M is the maximum coalition size and $|z'|$ the number of present features in instance z'. Lundberg and Lee show that linear regression with this kernel weight yields Shapley values. If you would use the SHAP kernel with LIME on the coalition data, LIME would also estimate Shapley values!

We can be a bit smarter about the sampling of coalitions: The smallest and largest coalitions take up most of the weight. We get better Shapley value estimates by using some of the sampling budget K to include these high-weight coalitions instead of sampling blindly. We start with all possible coalitions with 1 and M-1 features, which makes 2 times M coalitions in total. When we have enough budget left (current budget is K - 2M), we can include coalitions with two features and with M-2 features and so on. From the remaining coalition sizes, we sample with readjusted weights.

We have the data, the target and the weights. Everything to build our weighted linear regression model:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

We train the linear model g by optimizing the following loss function L:

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

where Z is the training data. This is the good old boring sum of squared errors that we usually optimize for linear models. The estimated coefficients of the model, the $\phi_j$'s are the Shapley values.

Since we are in a linear regression setting, we can also make use of the standard tools for regression. For example, we can add regularization terms to make the model sparse. If we add an L1 penalty to the loss L, we can create sparse explanations. (I am not so sure whether the resulting coefficients would still be valid Shapley values though)

## 5.10.3 TreeSHAP

Lundberg et. al (2018)[49] proposed TreeSHAP, a variant of SHAP for tree-based machine learning models such as decision trees, random forests and gradient boosted trees. TreeSHAP was introduced as a fast, model-specific alternative to KernelSHAP, but it turned out that it can produce unintuitive feature attributions.

TreeSHAP defines the value function using the conditional expectation $E_{X_S|X_C}(f(x)|x_S)$ instead of the marginal expectation. The problem with the conditional expectation is that features that have no influence on the prediction function f can get a TreeSHAP estimate different from zero.[50][51] The non-zero estimate can happen when the feature is correlated with another feature that actually has an influence on the prediction.

How much faster is TreeSHAP? Compared to exact KernelSHAP, it reduces the computational complexity from $O(TL2^M)$ to $O(TLD^2)$, where T is the number of trees, L is the maximum number of leaves in any tree and D the maximal depth of any tree.

TreeSHAP uses the conditional expectation $E_{X_S|X_C}(f(x)|x_S)$ to estimate effects. I will give you some intuition on how we can compute the expected prediction for a single tree, an instance x and feature subset S. If we conditioned on all features -- if S was the set of all features -- then the prediction from the node in which the instance x falls would be the expected prediction. If we did no condition on any feature -- if S was empty -- we would use the weighted average of predictions of all terminal nodes. If S contains some, but not all, features, we ignore predictions of unreachable nodes. Unreachable means that the decision path that leads to this node contradicts values in $x_S$. From the remaining terminal nodes, we average the predictions weighted by node sizes (i.e. number of training samples in that node). The mean of the remaining terminal nodes, weighted by the number of instances per node, is the expected prediction for x given S. The problem is that we have to apply this procedure for each possible subset S of the feature values. TreeSHAP computes in polynomial time instead of exponential. The basic idea is to push all possible subsets S down the tree at the same time. For each decision node we have to keep track of the number of subsets. This depends on the subsets in the parent node and the split feature. For example, when the first split in a tree is on feature x3, then all the subsets that contain feature x3 will go to one node (the one where x goes). Subsets that do not contain feature x3 go to both nodes with reduced weight. Unfortunately, subsets of different sizes have different weights. The algorithm has to keep track of the overall weight of the subsets in each node. This complicates the algorithm. I refer to the original paper for details of TreeSHAP. The computation can be expanded to more trees: Thanks to the Additivity property of Shapley values, the Shapley values of a tree ensemble is the (weighted) average of the Shapley values of the individual trees.

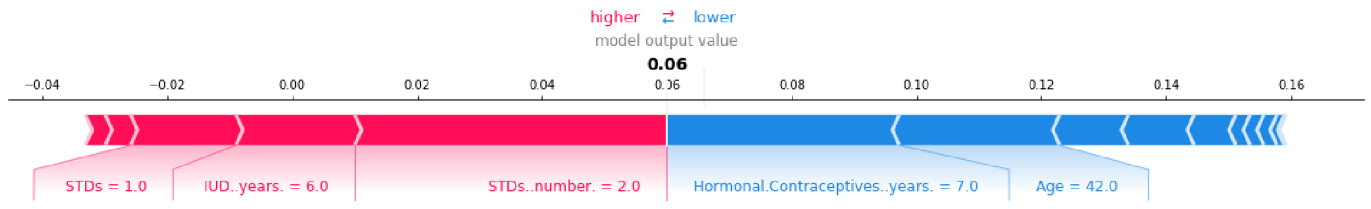
Next, we will look at SHAP explanations in action.

### 5.10.4 Examples

I trained a random forest classifier with 100 trees to predict the risk for cervical cancer. We will use SHAP to explain individual predictions. We can use the fast TreeSHAP estimation method instead of the slower KernelSHAP method, since a random forest is an ensemble of trees. But instead of relying on the conditional distribution, this example uses the marginal distribution. This is described in the package, but not in the original paper. The Python TreeSHAP function is slower with the marginal distribution, but still faster than KernelSHAP, since it scales linearly with the rows in the data.

Because we use the marginal distribution here, the interpretation is the same as in the Shapley value chapter. But with the Python shap package comes a different visualization: You can visualize feature attributions such as Shapley values as "forces". Each feature value is a force that either increases or decreases the prediction. The prediction starts from the baseline. The baseline for Shapley values is the average of all predictions. In the plot, each Shapley value is an arrow that pushes to increase (positive value) or decrease (negative value) the prediction. These forces balance each other out at the actual prediction of the data instance.

The following figure shows SHAP explanation force plots for two women from the cervical cancer dataset:



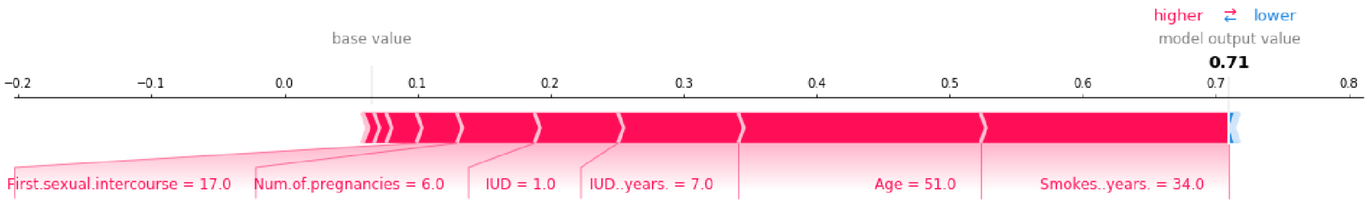


FIGURE 5.50: SHAP values to explain the predicted cancer probabilities of two individuals. The baseline -- the average predicted probability -- is 0.066. The first woman has a low predicted risk of 0.06. Risk increasing effects such as STDs are offset by decreasing effects such as age. The second woman has a high predicted risk of 0.71. Age of 51 and 34 years of smoking increase her predicted cancer risk.

These were explanations for individual predictions.

Shapley values can be combined into global explanations. If we run SHAP for every instance, we get a matrix of Shapley values. This matrix has one row per data instance and one column per feature. We can interpret the entire model by analyzing the Shapley values in this matrix.

We start with SHAP feature importance.

## 5.10.5 SHAP Feature Importance

The idea behind SHAP feature importance is simple: Features with large absolute Shapley values are important. Since we want the global importance, we sum the absolute Shapley values per feature across the data:

$$I_j=\sum_{i=1}^n|\phi_j^{(i)}|$$

Next, we sort the features by decreasing importance and plot them. The following figure shows the SHAP feature importance for the random forest trained before for predicting cervical cancer.

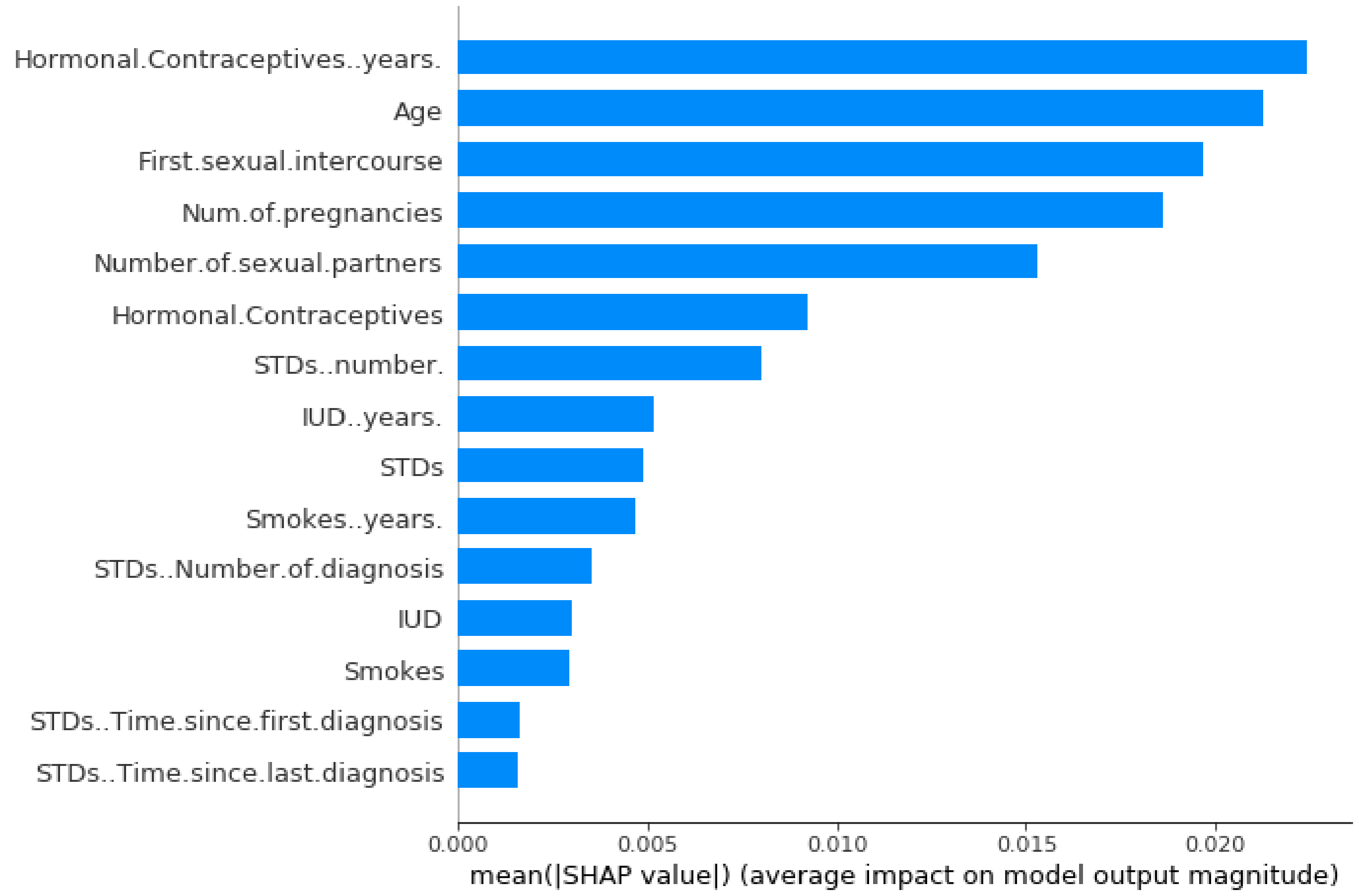


FIGURE 5.51: SHAP feature importance measured as the mean absolute Shapley values. The number of years with hormonal contraceptives was the most important feature, changing the predicted absolute cancer probability on average by 2.4 percentage points (0.024 on x-axis).

SHAP feature importance is an alternative to permutation feature importance. There is a big difference between both importance measures: Permutation feature importance is based on the decrease in model performance. SHAP is based on magnitude of feature attributions.

The feature importance plot is useful, but contains no information beyond the importances. For a more informative plot, we will next look at the summary plot.

### 5.10.6 SHAP Summary Plot

The summary plot combines feature importance with feature effects. Each point on the summary plot is a Shapley value for a feature and an instance. The position on the y-axis is determined by the feature and on the x-axis by the Shapley value. The color represents the value of the feature from low to high. Overlapping points are jittered in y-axis direction, so we get a sense of the distribution of the Shapley values per feature. The features are ordered according to their importance.

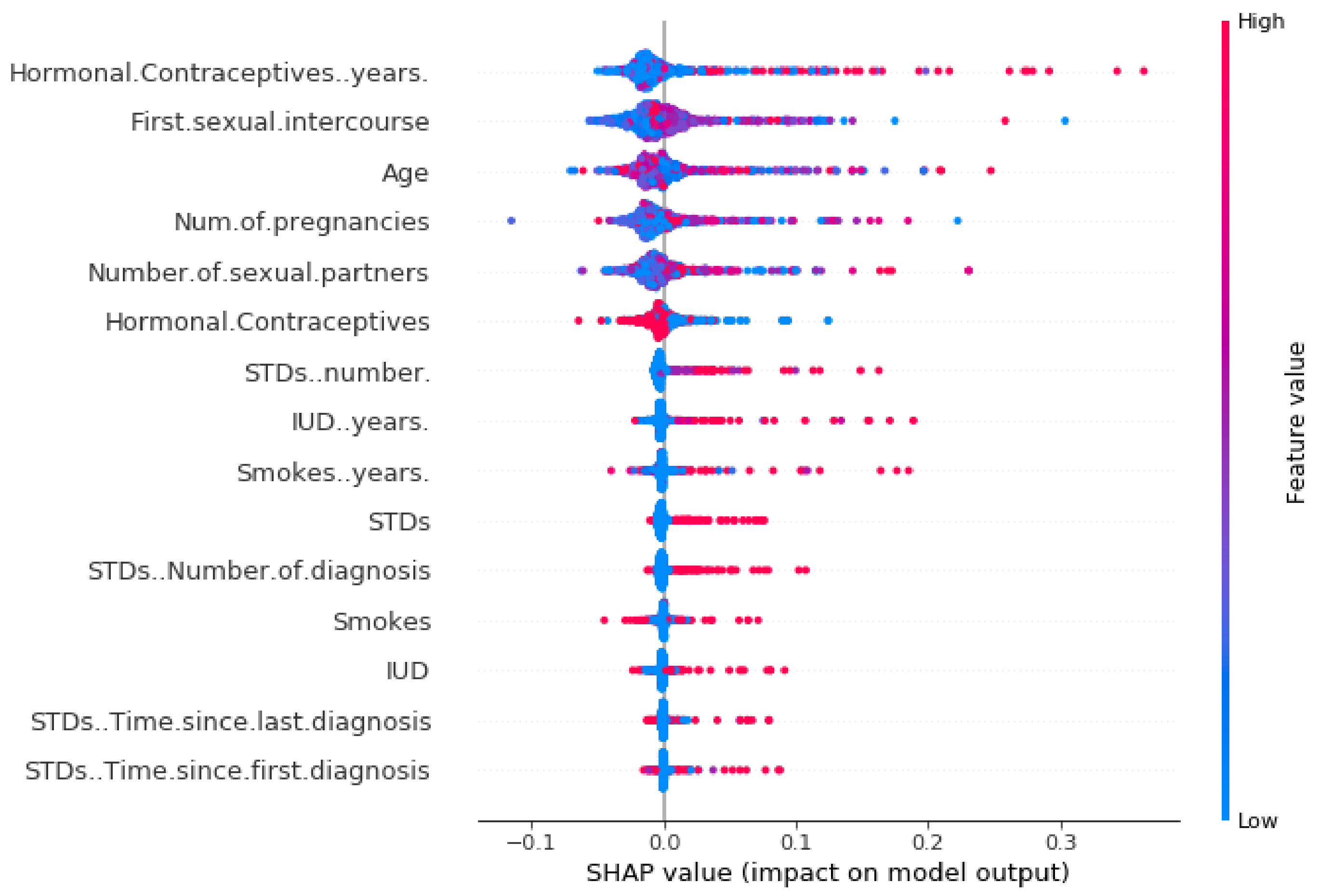


FIGURE 5.52: SHAP summary plot. Low number of years on hormonal contraceptives reduce the predicted cancer risk, a large number of years increases the risk. Your regular reminder: All effects describe the behavior of the model and are not necessarily causal in the real world.

In the summary plot, we see first indications of the relationship between the value of a feature and the impact on the prediction. But to see the exact form of the relationship, we have to look at SHAP dependence plots.

### 5.10.7 SHAP Dependence Plot

SHAP feature dependence might be the simplest global interpretation plot: 1) Pick a feature. 2) For each data instance, plot a point with the feature value on the x-axis and the corresponding Shapley value on the y-axis. 3) Done.

Mathematically, the plot contains the following points: $\{(x_j^{(i)}, \phi_j^{(i)})\}_{i=1}^n$

The following figure shows the SHAP feature dependence for years on hormonal contraceptives:

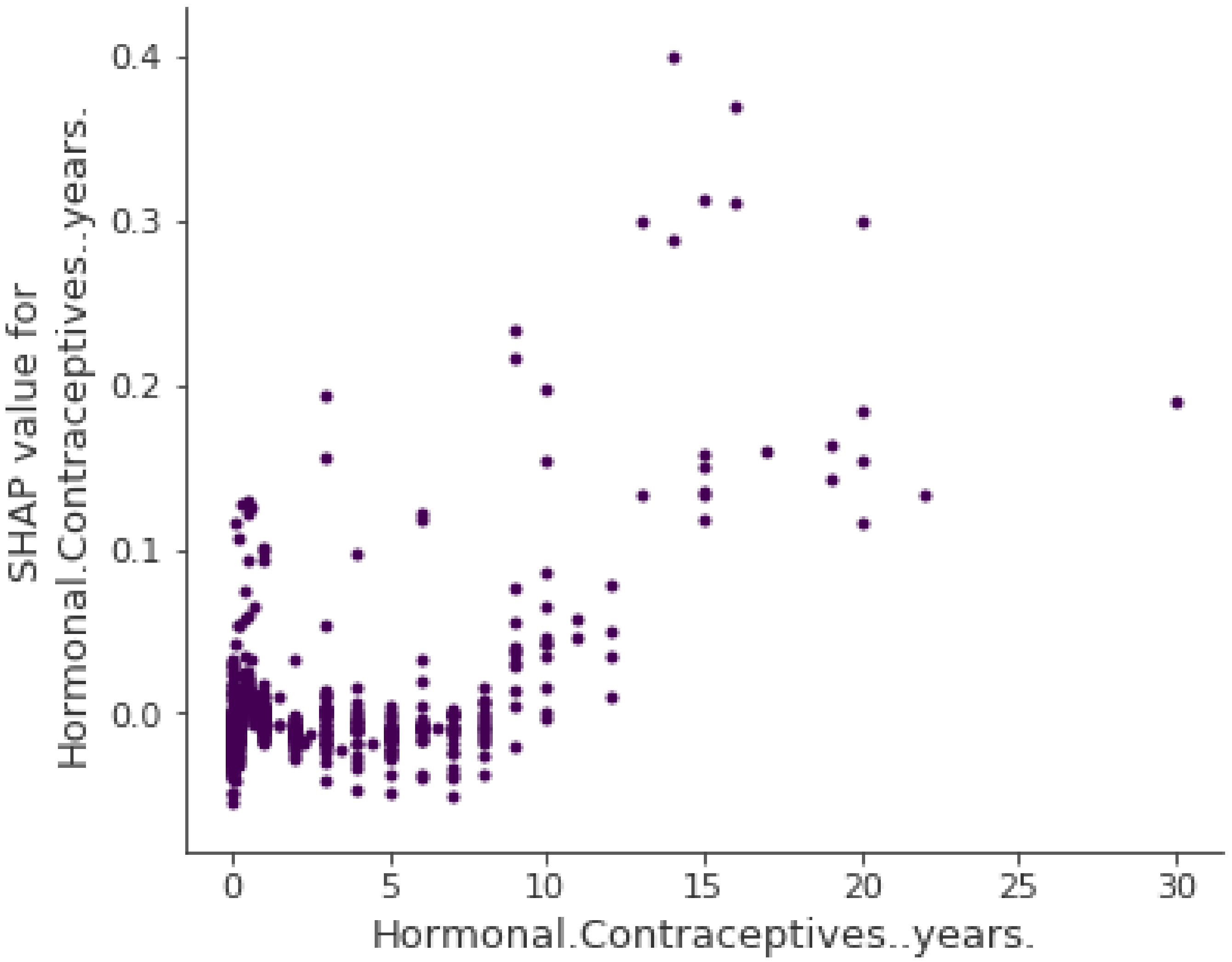


FIGURE 5.53: SHAP dependence plot for years on hormonal contraceptives. Compared to 0 years, a few years lower the predicted probability and a high number of years increases the predicted cancer probability.

SHAP dependence plots are an alternative to partial dependence plots and accumulated local effects. While PDP and ALE plot show average effects, SHAP dependence also shows the variance on the y-axis. Especially in case of interactions, the SHAP dependence plot will be much more dispersed in the y-axis. The dependence plot can be improved by highlighting these feature interactions.

### 5.10.8 SHAP Interaction Values

The interaction effect is the additional combined feature effect after accounting for the individual feature effects. The Shapley interaction index from game theory is defined as:

$$\phi_{i,j}=\sum_{S\subseteq\setminus\{i,j\}}\frac{|S|!(M-|S|-2)!}{2(M-1)!}\delta_{ij}(S)$$

when $i\neq j$ and:

$$\delta_{ij}(S)=f_x(S\cup\{i,j\})-f_x(S\cup\{i\})-f_x(S\cup\{j\})+f_x(S)$$

This formula subtracts the main effect of the features so that we get the pure interaction effect after accounting for the individual effects. We average the values over all possible feature coalitions S, as in the Shapley value computation. When we compute SHAP interaction values for all features, we get one matrix per instance with dimensions M x M, where M is the number of features.

How can we use the interaction index? For example, to automatically color the SHAP feature dependence plot with the strongest interaction:

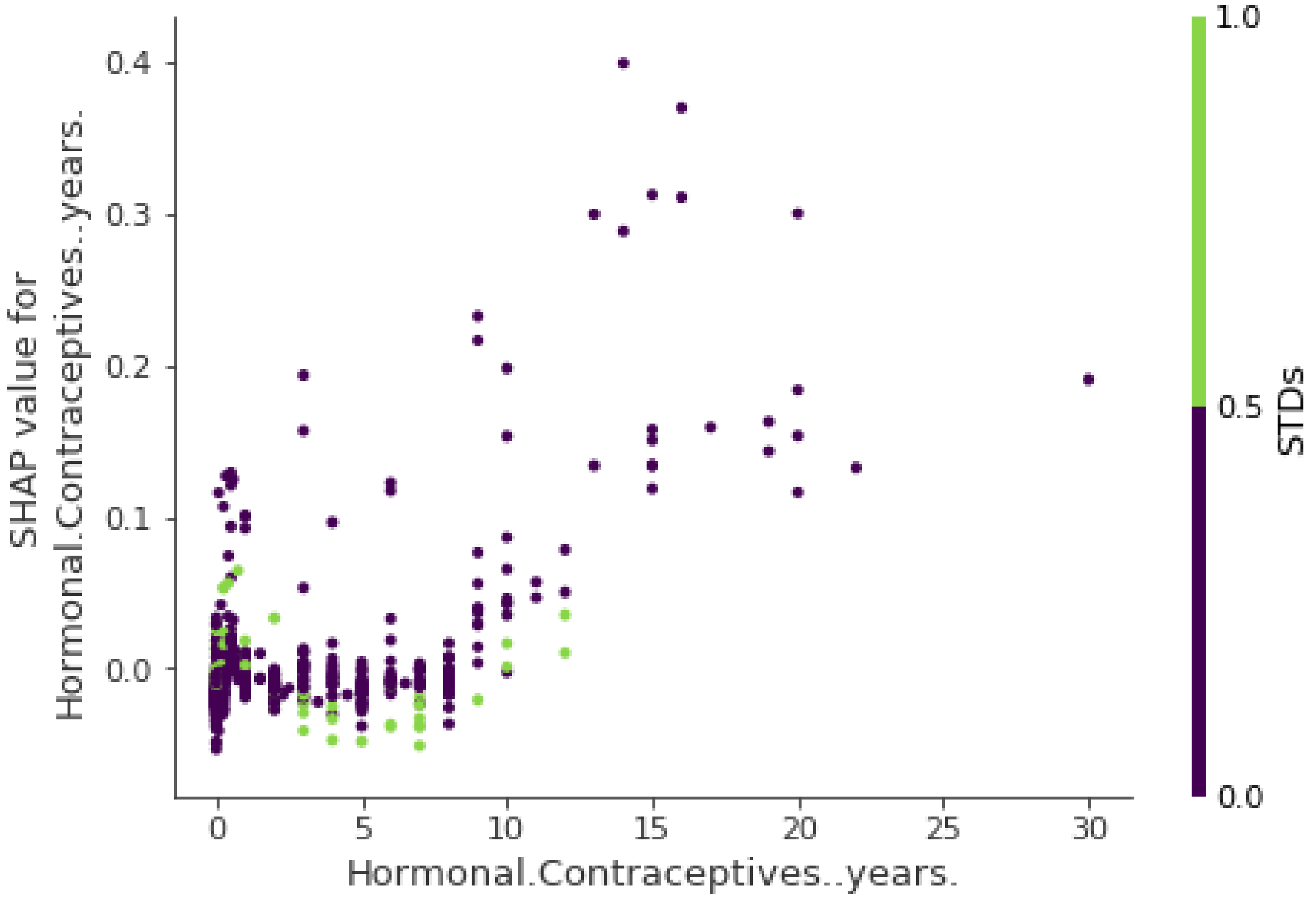


FIGURE 5.54: SHAP feature dependence plot with interaction visualization. Years on hormonal contraceptives interacts with STDs. In cases close to 0 years, the occurence of an STD increases the predicted cancer risk. For more years on contraceptives, the occurence of an STD reduces the predicted risk. Again, this is not a causal model. Effects might be due to confounding (e.g. STDs and lower cancer risk could be correlated with more doctor visits).

## 5.10.9 Clustering SHAP values

You can cluster your data with the help of Shapley values. The goal of clustering is to find groups of similar instances. Normally, clustering is based on features. Features are often on different scales. For example, height might be measured in meters, color intensity from 0 to 100 and some sensor output between -1 and 1. The difficulty is to compute distances between instances with such different, non-comparable features.

SHAP clustering works by clustering on Shapley values of each instance. This means that you cluster instances by explanation similarity. All SHAP values have the same unit -- the unit of the prediction space. You can use any clustering method. The following example uses hierarchical agglomerative clustering to order the instances.

The plot consists of many force plots, each of which explains the prediction of an instance. We rotate the force plots vertically and place them side by side according to their clustering similarity.

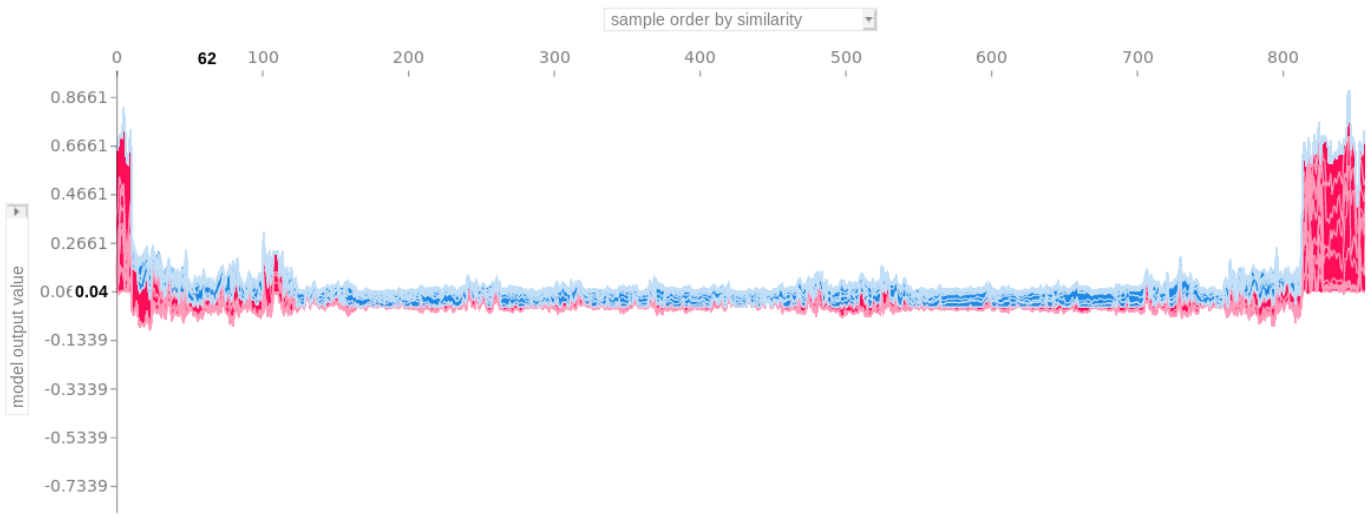


FIGURE 5.55: Stacked SHAP explanations clustered by explanation similarity. Each position on the x-axis is an instance of the data. Red SHAP values increase the prediction, blue values decrease it. A cluster stands out: On the right is a group with a high predicted cancer risk.

## 5.10.10 Advantages

Since SHAP computes Shapley values, all the advantages of Shapley values apply: SHAP has a **solid theoretical foundation** in game theory. The prediction is **fairly distributed** among the feature values. We get **contrastive explanations** that compare the prediction with the average prediction.

SHAP **connects LIME and Shapley values.** This is very useful to better understand both methods. It also helps to unify the field of interpretable machine learning.

SHAP has a **fast implementation for tree-based models.** I believe this was key to the popularity of SHAP, because the biggest barrier for adoption of Shapley values is the slow computation.

The fast computation makes it possible to compute the many Shapley values needed for the **global model interpretations.** The global interpretation methods include feature importance, feature dependence, interactions, clustering and summary plots. With SHAP, global interpretations are consistent with the local explanations, since the Shapley values are the "atomic unit" of the global interpretations. If you use LIME for local explanations and partial dependence plots plus permutation feature importance for global explanations, you lack a common foundation.

## 5.10.11 Disadvantages

**KernelSHAP is slow.** This makes KernelSHAP impractical to use when you want to compute Shapley values for many instances. Also all global SHAP methods such as SHAP feature importance require computing Shapley values for a lot of instances.

**KernelSHAP ignores feature dependence.** Most other permutation based interpretation methods have this problem. By replacing feature values with values from random instances, it is usually easier to randomly sample from the marginal distribution. However, if features are dependent, e.g. correlated, this leads to putting too much weight on unlikely data points. TreeSHAP solves this problem by explicitly modeling the conditional expected prediction.

**TreeSHAP can produce unintuitive feature attributions.** While TreeSHAP solves the problem of extrapolating to unlikely data points, it introduces a new problem. TreeSHAP changes the value function by relying on the conditional expected prediction. With the change in the value function, features that have no influence on the prediction can get a TreeSHAP value different from zero.

The disadvantages of Shapley values also apply to SHAP: Shapley values **can be misinterpreted** and access to data is needed to compute them for new data (except for TreeSHAP).

It is possible to create intentionally misleading interpretations with SHAP, which can hide biases [52]. If you are the data scientist creating the explanations, this is not an actual problem (it would even be an advantage if you are the evil data scientist who wants to create misleading explanations). It is a disadvantage as the receiver of an explanation, as you can be less sure about their truthfulness.

### 5.10.12 Software

The authors implemented SHAP in the shap Python package. This implementation works for tree-based models in the scikit-learn machine learning library for Python. The shap package was also used for the examples in this chapter. SHAP is integrated into the tree boosting frameworks xgboost and LightGBM. In R, there is the shapper and fastshap packages. SHAP is also included in the R xgboost package.

48. Lundberg, Scott M., and Su-In Lee. "A unified approach to interpreting model predictions." Advances in Neural Information Processing Systems. 2017.↩

49. Lundberg, Scott M., Gabriel G. Erion, and Su-In Lee. "Consistent individualized feature attribution for tree ensembles." arXiv preprint arXiv:1802.03888 (2018).↩

50. Sundararajan, Mukund, and Amir Najmi. "The many Shapley values for model explanation." arXiv preprint arXiv:1908.08474 (2019).↩

51. Janzing, Dominik, Lenon Minorics, and Patrick Blöbaum. "Feature relevance quantification in explainable AI: A causality problem." arXiv preprint arXiv:1910.13413 (2019).↩

52. Slack, Dylan, et al. "Fooling lime and shap: Adversarial attacks on post hoc explanation methods." Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society. 2020.↩