8.6 SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2017)[42] 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.

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

8.6.1 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 superpixels 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 superpixels. 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

[42] Lundberg, Scott M., and Su-In Lee. "A unified approach to interpreting model predictions." Advances in Neural Information Processing Systems (2017).

SHAP多了 kernel-based estimation 、TreeSHAP aggregation of Shapley value

KernelSHAP 和 Tree SHAP 都是另一种 Shapley value 的估算方式

Kernel SHAP 受 LIME 的启发

SHAP 可以将 shapley value 解释成可相加的归因方法，一个线性 model

SHAP 的解释
M是最大的组合大小
z'是组合向量
ϕ_j是feature j的归因值
在组合向量中，1表示该feature的值是存在的；0表示不存在

you know about Shapley values. To compute Shapley values, we simulate that only some feature 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 ϕ'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 ϕ'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

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

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.

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

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"[43]. A missing feature

[43] https://github.com/slundberg/shap/issues/175#issuecomment-407134438

discrepancy 差异

E_X = expectation operator is defined exclusively over the X

Missingness : 当 feature 没有参与的时候，它的归因是0

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 $\hat{f}_x(z') = \hat{f}(h_x(z'))$ and z'_j indicate that $z'_j = 0$. For any two models f and f' that satisfy:

$$\hat{f}'_x(z') - \hat{f}'_x(z'_j) \geq \hat{f}_x(z') - \hat{f}_x(z'_j)$$

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

$$\phi_j(\hat{f}', x) \geq \phi_j(\hat{f}, x)$$

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.

8.6.2 KernelSHAP

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

- Sample coalitions $z'_k \in \{0,1\}^M, \quad k \in \{1,\dots,K\}$ (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 $\hat{f} : \hat{f}(h_x(z'_k))$
- Compute the weight for each z'_k with the SHAP kernel.
- Fit weighted linear model.
- Return Shapley values ϕ_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 cannot make predictions for them.") To get from coalitions of feature values to valid data instances, we need a function $h_x(z') = z$ where $h_x : \{0,1\}^M \to \mathbb{R}^p$. The function h_x maps 1's to the corresponding value from the instance x that we want to explain. For tabular

enforce 迫使

Consistency 一致性

如果 model 变了，那么 feature 的边缘贡献也会升高或保持不变。

1. 挑选K个向量组合
2. 将每个向量组合转化到原 feature space 后作预测。
3. 用 SHAP kernel 计算权重。
4. 训练加权线性 model
5. 从线性 model 中获回系数作为 Shapley value ϕ_k

h_x 是将组合向量映射为一个实例的方法，其中1代表这个 feature 值来自我们想要解释的实例，

对于表数据，0表示该 feature 我们从其它实例中随机取

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:

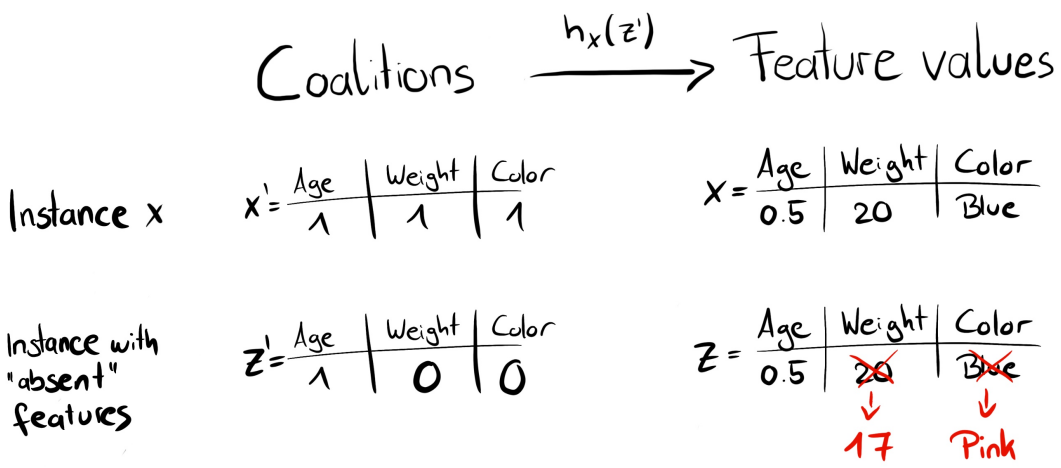

Figure 8.22: Function h_x maps a coalition to a valid instance. For present features (1), h_x maps to the feature values of x. For absent features (0), h_x maps to the values of a randomly sampled data instance.

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

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

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. The solution would be to sample from the conditional distribution, which changes the value function, and therefore the game to which Shapley values are the solution. As a result, the Shapley values have a different interpretation: For example, a feature that might not have been used by the model at all can have a non-zero Shapley value when the conditional sampling is used. For the marginal game, this feature value would always get a Shapley value of 0, because otherwise it would violate the Dummy axiom.

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

这里将"feature的不存在"和"feature取其它实例值"划上等号

它是permutation-based

KernelSHAP的问题：会受到失真实例的影响

解决方法是从条件分布中抽样

于是Shapley value可以另作解释：在条件抽样时，没有被model所使用的feature可以有非零shapley value，而边缘分布下，该feature的shapley value必须为0.

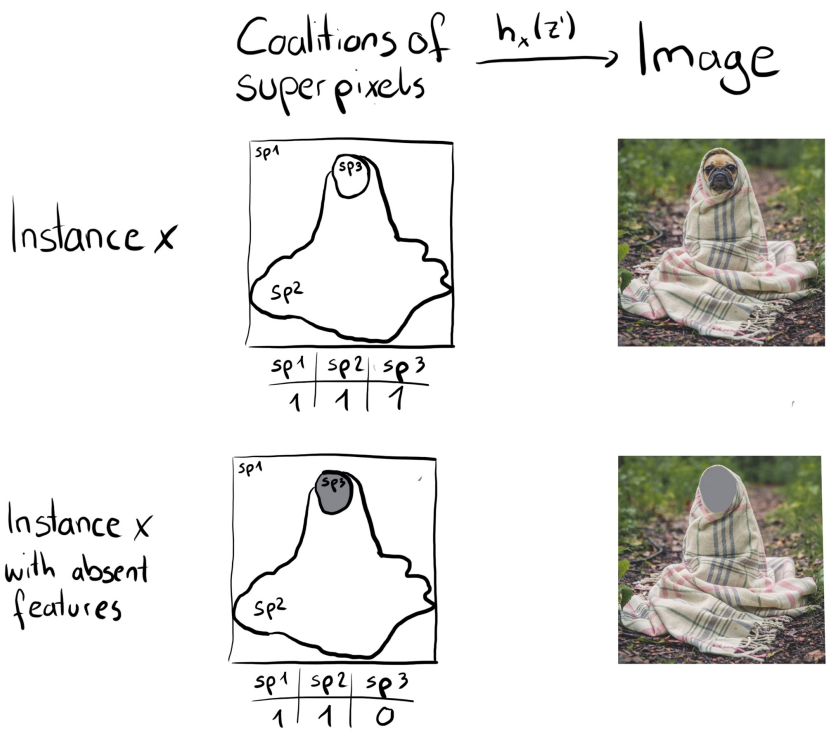

Figure 8.23: Function h_x maps coalitions of superpixels (sp) to images. Superpixels are groups of pixels. For present features (1), h_x returns the corresponding part of the original image. For absent features (0), h_x greys out the corresponding area. Assigning the average color of surrounding pixels 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 this feature's isolated main effect on the prediction. If a coalition consists of all but one feature, we can learn about this feature's total effect (main effect plus feature interactions). If a coalition consists of half the features, we learn little about an individual feature's contribution, as there are many possible coalitions with half of the features. To achieve Shapley compliant

和LIME的最大不同是权重过程.

LIME的权重取决于它们有多接近源实例.

SHAP 权重取决于组合中能估算的 Shapley value.

少1和多1的组合获得最大的权重.

① 只有单个 feature = {1}

② 所有features但除了那一个 {0111}

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 2 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 we need 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:

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

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 ϕ_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.)

SHAP 核心

$\binom{M}{|z'|}$ 是什么 = binomial coefficient

二项式系数

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$

最大和最小的组合占了最多的权重

一个很好的抽样方式：

从最小组合数和最大组合数开始

找到最佳的 loss：

[实际预测 − 加权线性回归]^2 SHAP Kernel

regularization term 正则项

Sparse 稀少的；简介的

8.6.3 TreeSHAP

Lundberg et al. (2018)[44] 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 $E_{X_S|X_C}(\hat{f}(x)|x_S)$ 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 as shown by Sundararajan et al. (2019) [45] and Janzing et al. (2019) [46]. 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 $E_{X_S|X_C}(\hat{f}(x)|x_S)$ 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 would not condition the prediction 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

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

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

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

TreeSHAP是 model-specific alternative to KernelSHAP
它比 KernelSHAP 快，但它会产生非直觉的 feature 归因
因为它使用条件分布，

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.

8.6.4 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:

Fig 8.24

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.

Shapley value 可以组成全局解释。

We start with SHAP feature importance.

8.6.5 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 average the **absolute** Shapley values per feature across the data:

Shapley value 绝对值的平均值可以被解释为 global importance

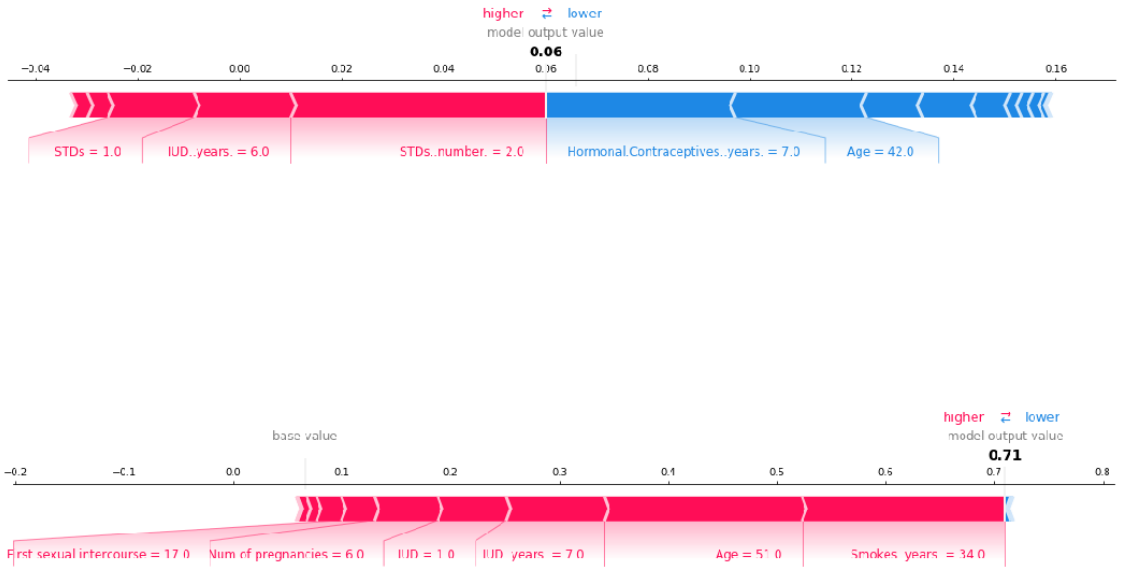

Figure 8.24: SHAP values to explain the predicted cancer probabilities of two individuals. The baseline – the average predicted probability – is 0.066. The first woman has a low predicted risk of 0.06. Risk increasing effects such as STDs are offset by decreasing effects such as age. The second woman has a high predicted risk of 0.71. Age of 51 and 34 years of smoking increase her predicted cancer risk.

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

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.

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.

magnitude 大小；量级；

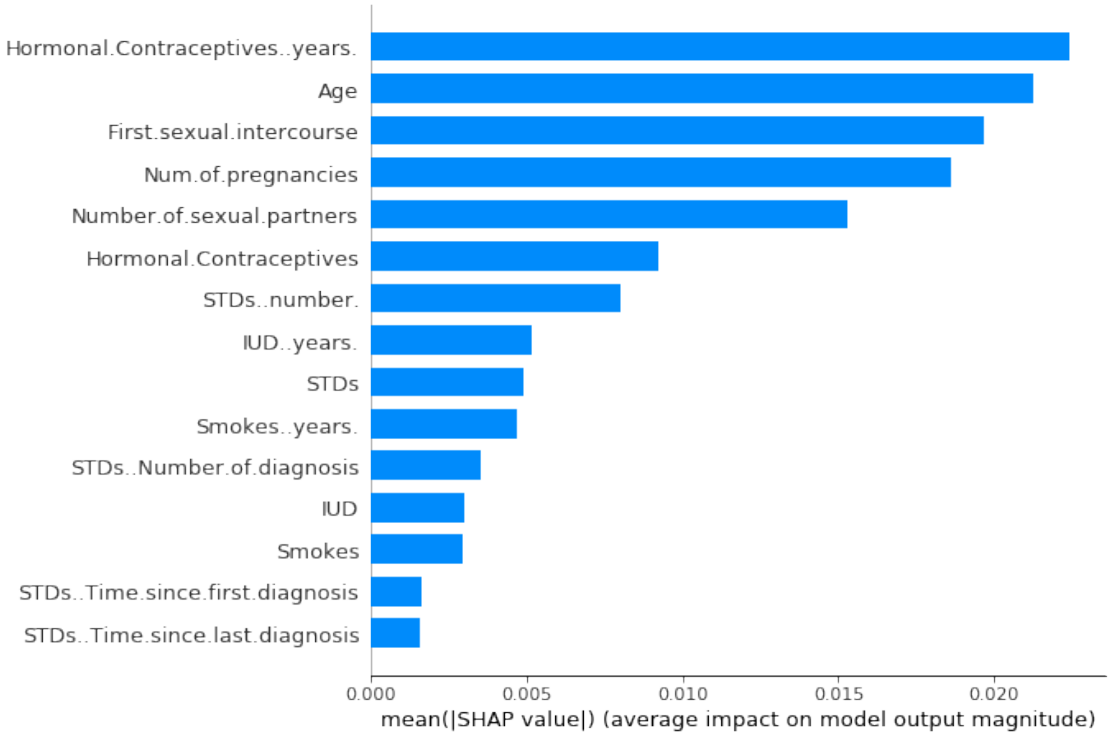

Figure 8.25: SHAP feature importance measured as the mean absolute Shapley values. The number of years with hormonal contraceptives was the most important feature, changing the predicted absolute cancer probability on average by 2.4 percentage points (0.024 on x-axis).

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.

8.6.6 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.

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.

jitter 晃动

SHAP 总结图

图上的每一个点是一个实例在一个 feature 上的 sv

Y轴 取决于 feature

X轴是 sv

重叠的 sv 在 Y轴上晃动

和 violin 图差不多，用于体现分布

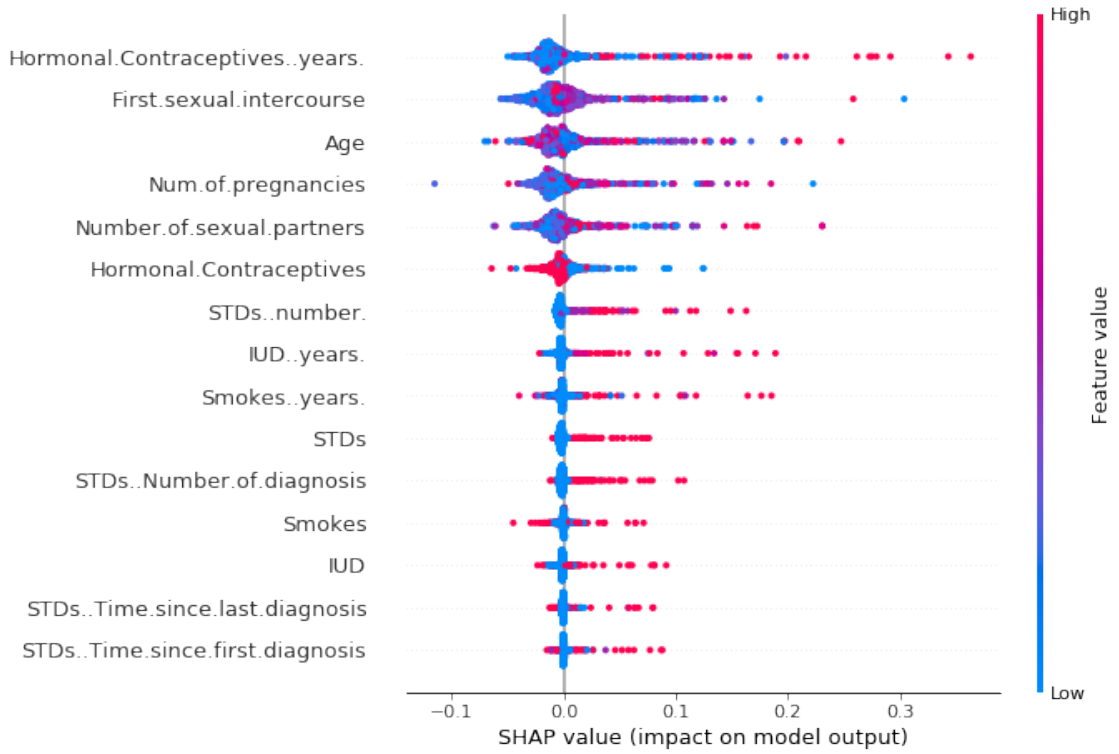

Figure 8.26: SHAP summary plot. Low number of years on hormonal contraceptives reduce the predicted cancer risk, a large number of years increases the risk. Your regular reminder: All effects describe the behavior of the model and are not necessarily causal in the real world.

8.6.7 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:

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

SHAP 依赖图 是PDP和 ALE Plot 的替代

x 轴是 feature value

Y 轴是 SV

→ Fig 8.27

它还体现了分布

同时 SHAP 依赖图也可以表现出 feature 交互

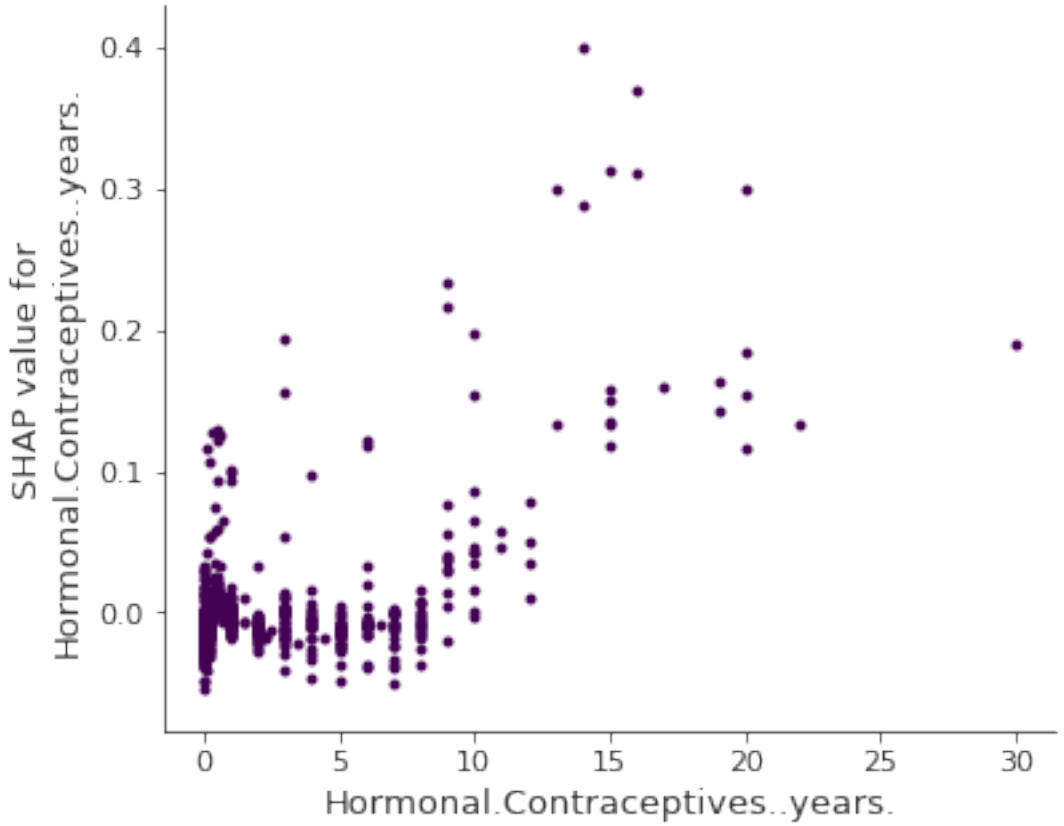

Figure 8.27: SHAP dependence plot for years on hormonal contraceptives. Compared to 0 years, a few years lower the predicted probability and a high number of years increases the predicted cancer probability.

will be much more dispersed in the y-axis. The dependence plot can be improved by highlighting these feature interactions.

8.6.8 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:

Shapley interaction

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

when $i \neq j$ and:

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

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:

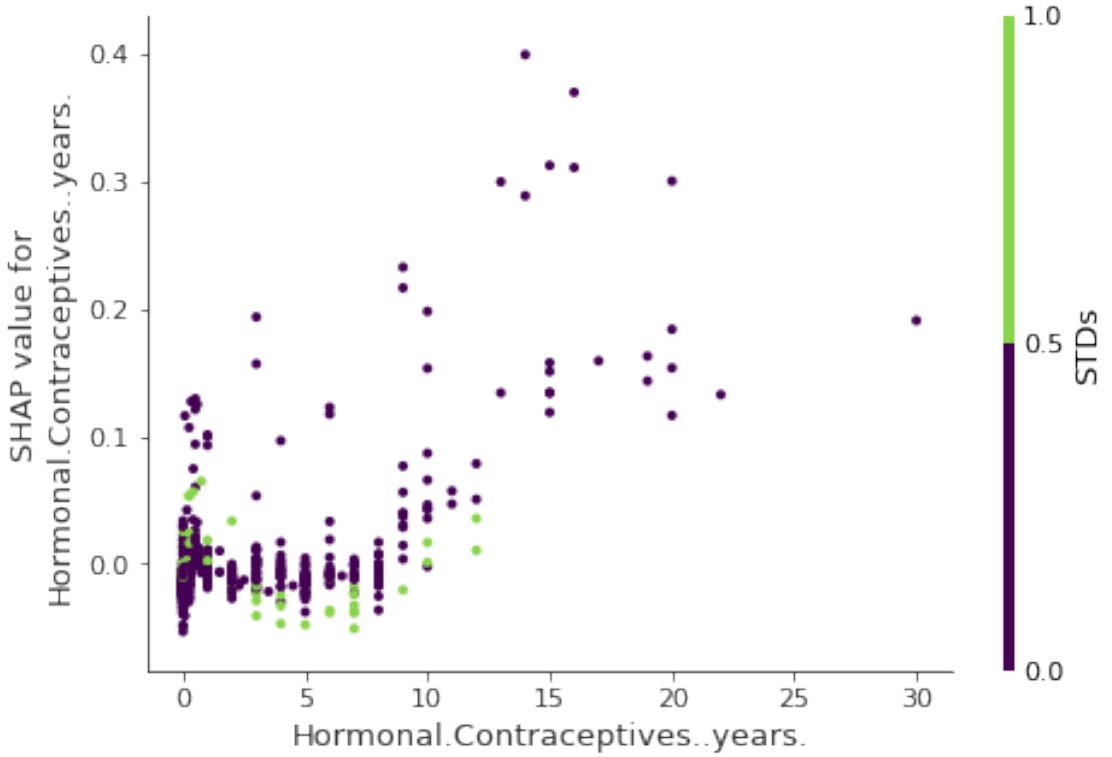

Figure 8.28: SHAP feature dependence plot with interaction visualization. Years on hormonal contraceptives interacts with STDs. In cases close to 0 years, the occurence of a STD increases the predicted cancer risk. For more years on contraceptives, the occurence of a STD reduces the predicted risk. Again, 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).

8.6.9 Clustering Shapley 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 the 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.

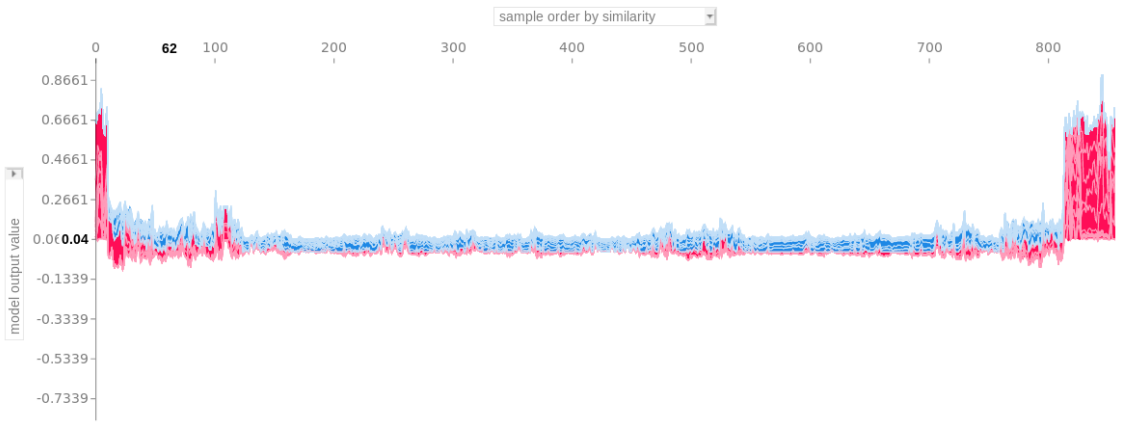

Figure 8.29: Stacked SHAP explanations clustered by explanation similarity. Each position on the x-axis is an instance of the data. Red SHAP values increase the prediction, blue values decrease it. One cluster stands out: On the right is a group with a high predicted cancer risk.

8.6.10 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.

Contrastive 对比的

优：

理论基础

和平均预测作对比解释

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.

8.6.11 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 does so by changing the value function and therefore slightly changes the game. 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 [47]. If you are the data scientist creating the explanations, this is not an

[47] Slack, Dylan, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. "Fooling lime and shap: Adversarial attacks on post hoc explanation methods." In Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp. 180-186 (2020).

结合了 Shapley value 和 LIME

对于 Tree-based 有高效实现

↓

计算迅速，对于将单独 SV 聚合成全局解释来说很有优势

缺点：

KernelSHAP 很慢

受失真实例的影响

TreeSHAP 求解了这个问题，它在条件分布下计算

但因此 TreeSHAP 会产生非直觉的 feature 归因

不参与预测的 feature 可能会有非零归因

actual problem (it would even be an advantage if you are the evil data scientist who wants to create misleading explanations). For the receivers of a SHAP explanation, it is a disadvantage: they cannot be sure about the truthfulness of the explanation.

8.6.12 Software

The authors implemented SHAP in the shap[48] Python package. This implementation works for tree-based models in the scikit-learn[49] 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[50] and LightGBM[51]. In R, there are the shapper[52] and fastshap[53] packages. SHAP is also included in the R xgboost[54] package.

[48] https://github.com/slundberg/shap
[49] https://scikit-learn.org/stable/
[50] https://github.com/dmlc/xgboost/tree/master/python-package
[51] https://github.com/microsoft/LightGBM
[52] https://modeloriented.github.io/shapper/
[53] https://github.com/bgreenwell/fastshap
[54] https://rdrr.io/cran/xgboost/man/xgb.plot.shap.html