Index:

Types of XAI Tasks:

1. Model Agnostic Methods:

- Global methods describe how features affect the prediction **on average**.
- In contrast, local methods aim to explain **individual predictions**.

2. Model Specific Methods:

- Tools that only work for the interpretation of e.g. neural networks are model-specific.

Model Agnostic Methods:

1. Partial Dependence Plots:

- A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic or more complex.
- **Cons:**
  It is assumed that the **feature(s) for which the partial dependence is computed are not correlated** with other features.It is the main disadvantage.
- **Implementation:**
  There are multiple packages and libraries that we can use to plot PDPs. If you are using R, there are packages including **iml**, **pdp** and **DALEX** for PDPs. For Python, the **PDPBox** package and the **PartialDependenceDisplay** function in the **sklearn.inspection** module are the best ones.

2. ALE Plots:

- ALE does have a competitive edge over partial dependence plot (PD) because it addresses the bias that arises in PD when the feature of interest is highly correlated with other features
- Example
  1.**ALE for two features only shows the additional interaction effect of the two features**

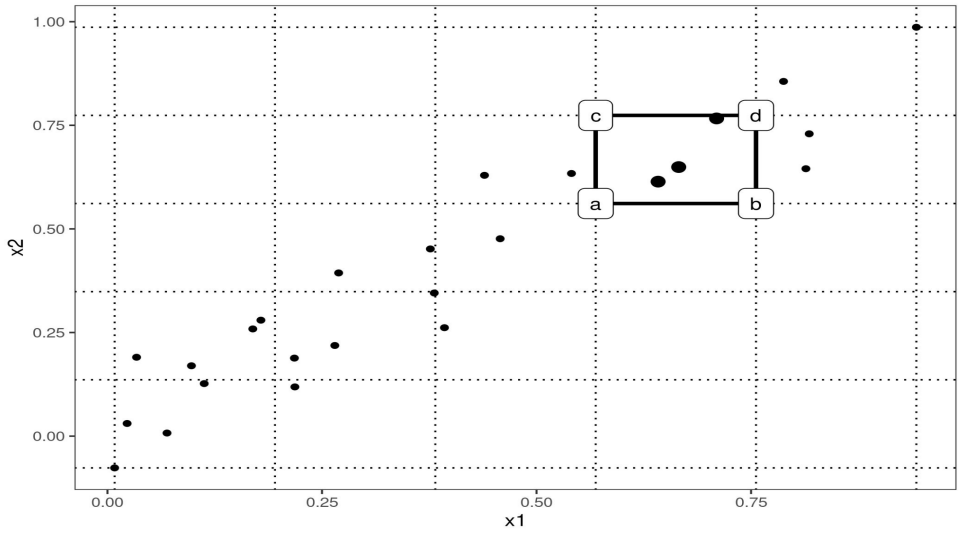
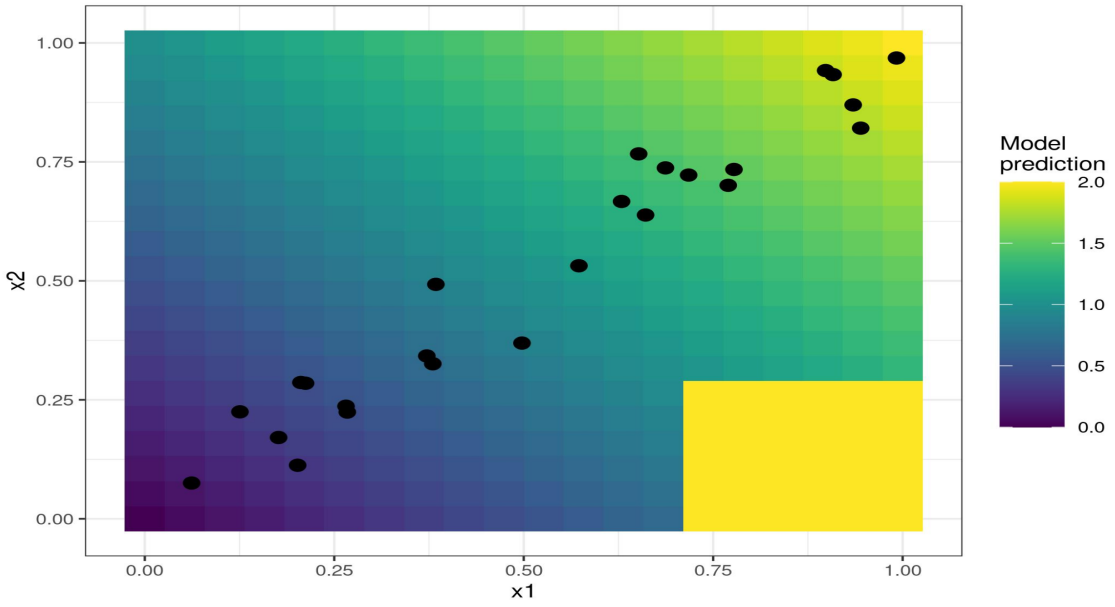


Figure:
Calculation of 2D-ALE. We place a grid over the two features. In each grid cell we calculate the 2nd-order differences for all instance within. We first replace values of x1 and x2 with the values from the cell corners. If a, b, c and d represent the "corner"-predictions of a manipulated instance (as labeled in the graphic), then

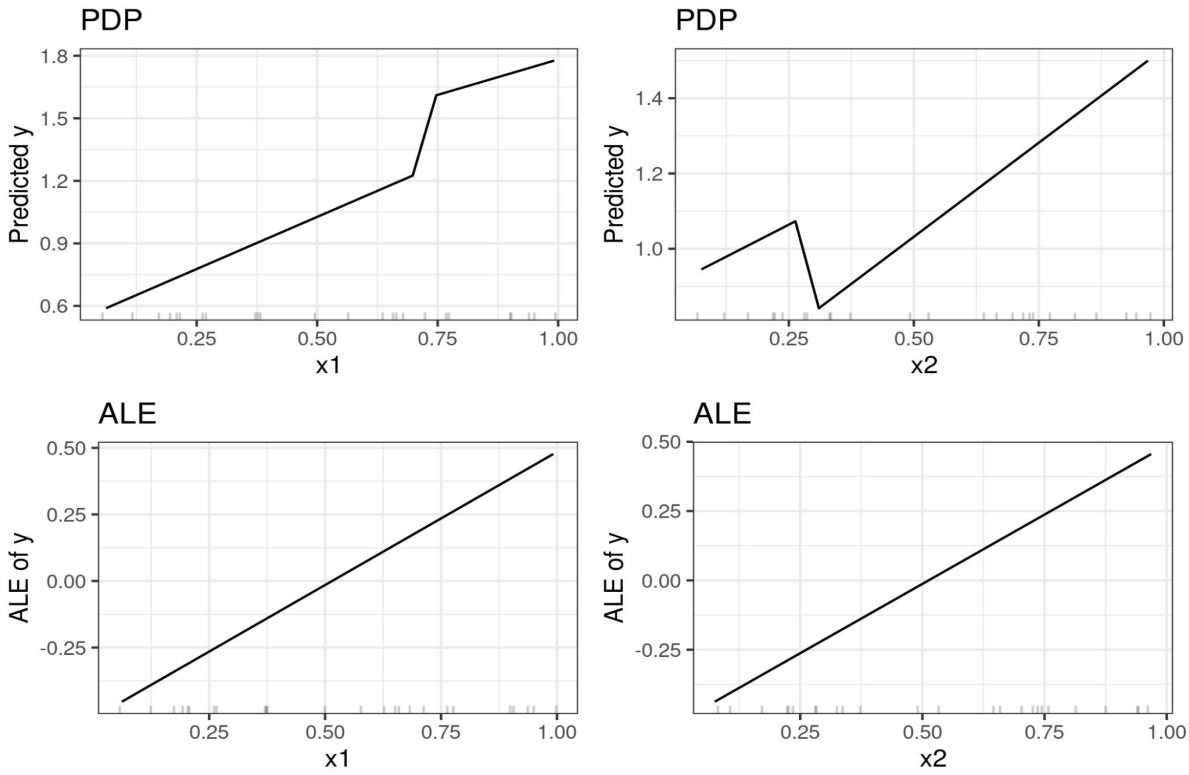
the 2nd-order difference is (d - c) - (b - a). The mean 2nd-order difference in each cell is accumulated over the grid and centered.

2. Two features and the predicted outcome. The model predicts the sum of the two features (shaded background), with the exception that if x1 is greater than 0.7 and x2 less than 0.3, the model always predicts 2. This area is far from the distribution of data (point cloud) and does not affect the performance of the model and also should not affect its interpretation.



- **Cons:**

  The interpretation also is usually limited within the “window” or “interval” defined. If the features are highly correlated (which is often the case), interpreting an effect across intervals is impossible.
- **Tools For Implementation: iml package,ALEPlot** package for R,**ALEPython package, Alibi package** for python
- **Example of PDP vs ALE:**
- The PDP estimates are influenced by the odd behavior of the model outside the data distribution (steep jumps in the plots). The ALE plots correctly identify that the machine learning model has a linear relationship between features and prediction, ignoring areas without data.



3. Feature Interaction:

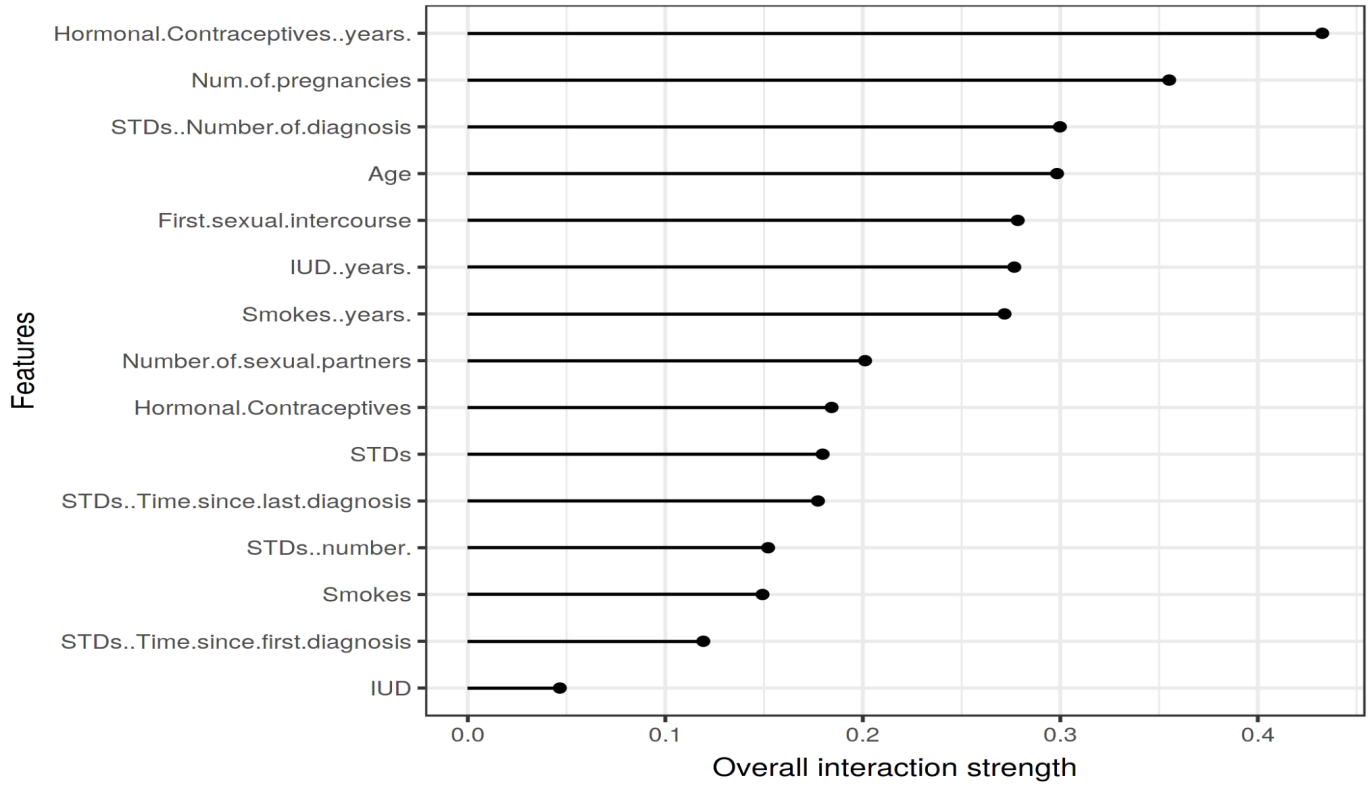
- When features interact with each other in a prediction model, the prediction cannot be expressed as the sum of the feature effects, because the effect of one feature depends on the value of the other feature.

  Mathematically, the H-statistic proposed by Friedman and Popescu for the interaction between feature j and k is:

$$H_{jk}^2 = \frac{\sum_{i=1}^{n} \left[PD_{jk}(x_j^{(i)}, x_k^{(i)}) - PD_j(x_j^{(i)}) - PD_k(x_k^{(i)}) \right]^2}{\sum_{i=1}^{n} PD_{jk}^2(x_j^{(i)}, x_k^{(i)})}$$

- **Example:**

  The interaction strength (H-statistic) for each feature with all other features for a random forest predicting the probability of cervical cancer. The years on hormonal contraceptives has the highest relative interaction effect with all other features, followed by the number of pregnancies.



- **Cons:**
- The computation involves estimating marginal distributions. These estimates also have a certain variance if we do not use all data points. **This means that as we sample points, the estimates also vary** from run to run and the results can be unstable.
- **If the features correlate strongly**, the assumption is violated and we integrate over feature combinations that are very unlikely in reality.

Tools For Implementation

- A popular approach used to visualize the interaction effects is the partial dependence plot (PDP) (using pdpbox) . We can use PDP to visualize how 2 features vary with the predictions. Variants like ICE and shapley also help you visualize the interaction effects in a similar manner.
- https://github.com/christophM/iml/tree/main/R

4. Functional Decomposition

$$\begin{aligned}\hat{f}(x) = &\hat{f}_0 + \hat{f}_1(x_1) + \ldots + \hat{f}_p(x_p) \\ &+ \hat{f}_{1,2}(x_1, x_2) + \ldots + \hat{f}_{1,p}(x_1, x_p) + \ldots + \hat{f}_{p-1,p}(x_{p-1}, x_p) \\ &+ \ldots \\ &+ \hat{f}_{1,\ldots,p}(x_1, \ldots, x_p)\end{aligned}$$

- **Example:**

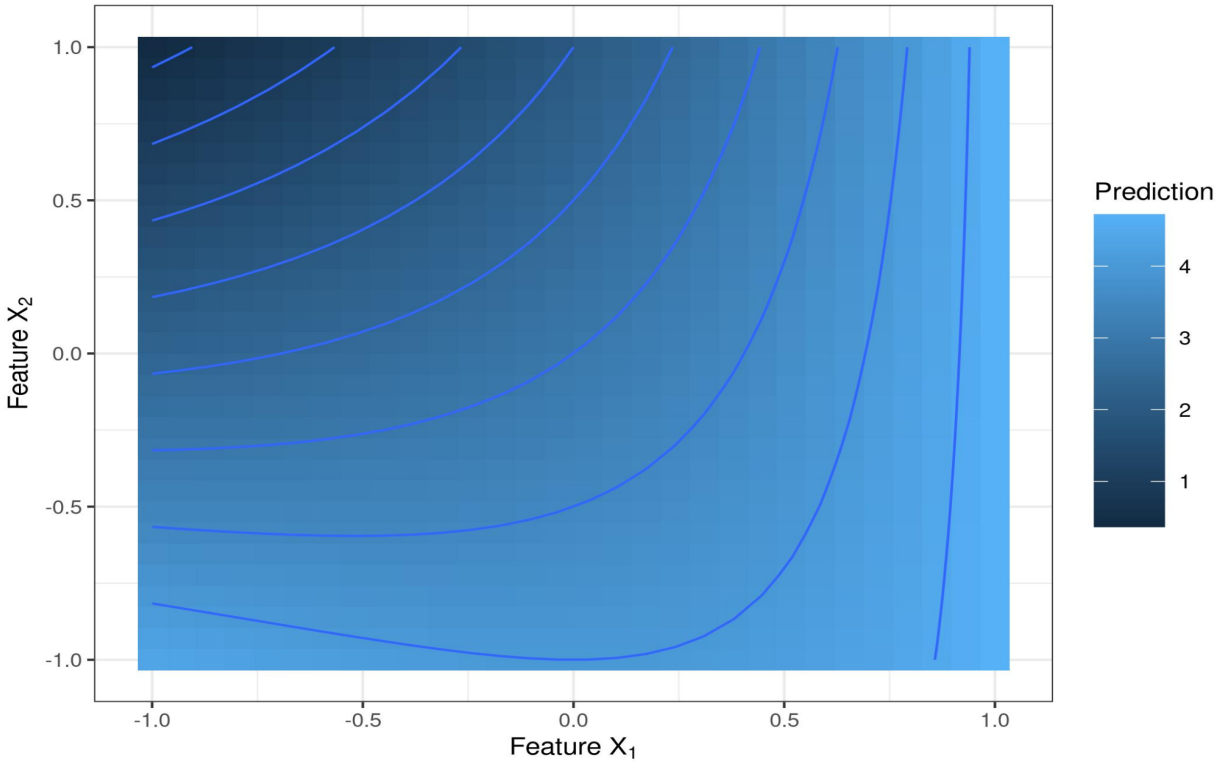


FIGURE 8.22: Prediction surface of a function with two features X1 & X2.

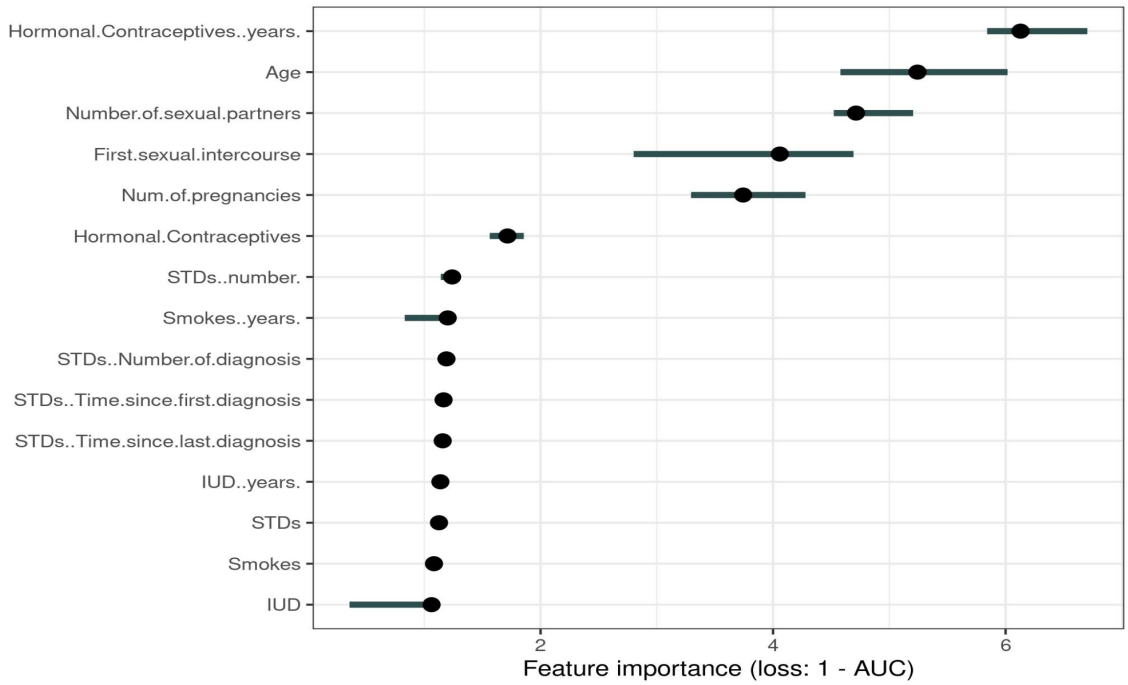
- The function takes large values when X1 is large and X2 is small, and it takes small values for large X2 and small X1 The prediction function is not simply an additive effect between the two features, but an interaction between the two. The presence of an interaction can be seen in the figure – the effect of changing values for feature X1 depends on the value that feature X2 has.
- In this tutorial, there are three methods that approach the functional decomposition in different ways:
    1. (Generalized) Functional ANOVA
    2. Accumulated Local Effects
    3. Statistical regression models
- **Cons**

- The concept of functional decomposition quickly reaches its limits for high-dimensional components beyond interactions between two features.
- Not only does this exponential explosion in the number of features limit practicability, since we cannot easily visualize higher-order interactions, but computational time is insane if we were to compute all interactions.
- The bottom-up approach – constructing regression models – is a quite manual process and imposes many constraints on the model that can affect predictive performance.
- Functional ANOVA requires independent features. Generalized functional ANOVA is very difficult to estimate. Accumulated local effect plots do not provide a variance decomposition.

5. Permutation Feature Importance

- Permutation feature importance measures the increase in the prediction error of the model after we permute the feature's values, which breaks the relationship between the feature and the true outcome.
- **Example:**

1. The importance of each of the features for predicting cervical cancer with a random forest. The most important feature was Hormonal.Contraceptives..years.. Permuting Hormonal.Contraceptives..years. resulted in an increase in 1-AUC by a factor of 6.13



Cons:

- Suppose you want to find out how robust your model's output is when someone manipulates the features. In this case, you would not be interested in how much the model performance decreases when a feature is permuted, **but how much of the model's output variance is explained by each feature. Model variance (explained by the features) and feature importance correlate strongly when the model generalizes well (i.e. it does not overfit).**
- The permutation feature importance depends on shuffling the feature, which adds randomness to the measurement. When the permutation is repeated, the **results might vary greatly**. Repeating the permutation and averaging the importance measures over repetitions stabilizes the measure, but increases the time of computation.
- If features are correlated, the permutation feature importance **can be biased by unrealistic data instances**. The problem is the same as with partial dependence plots:
- **Adding a correlated feature can decrease the importance of the associated feature** by splitting the importance between both features.

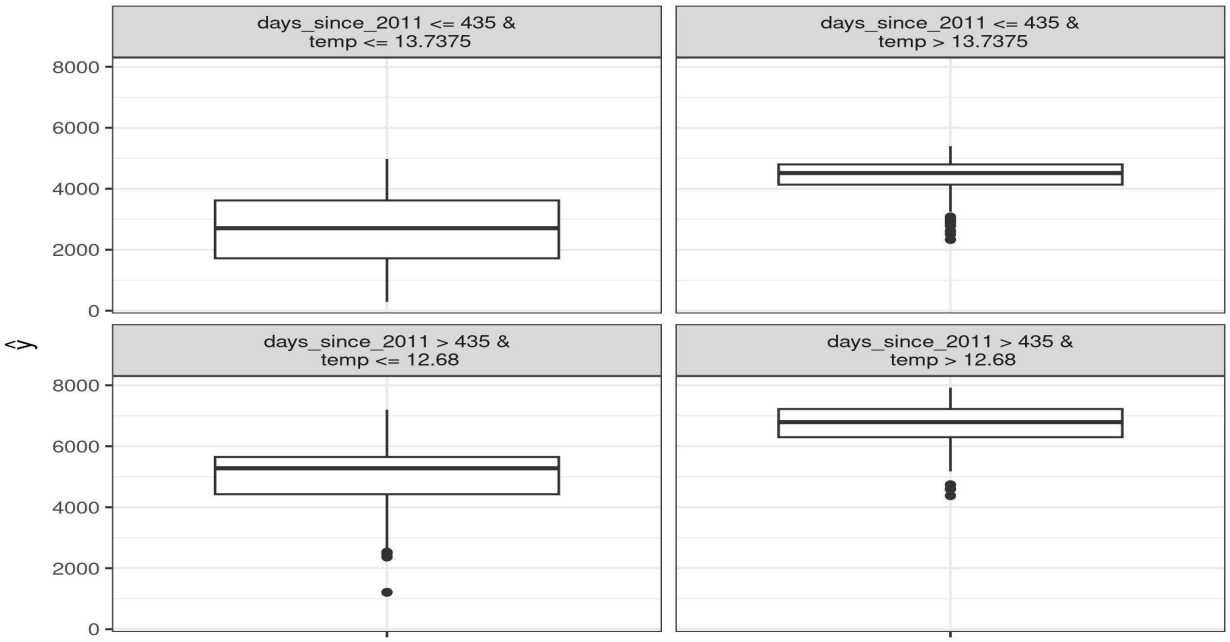
6. Global Surrogate Models:

- Perform the following steps to obtain a surrogate model:

1. Select a dataset X. This can be the same dataset that was used for training the black box model or a new dataset from the same distribution. You could even select a subset of the data or a grid of points, depending on your application.
2. For the selected dataset X, get the predictions of the black box model.
3. Select an interpretable model type (linear model, decision tree, …).
4. Train the interpretable model on the dataset X and its predictions.
5. Congratulations! You now have a surrogate model.
6. Measure how well the surrogate model replicates the predictions of the black box model.
7. Interpret the surrogate model.

- **Example**

1. First, we train a support vector machine to predict the daily number of rented bikes given weather and calendar information. The support vector machine is not very interpretable, so we train a surrogate with a CART decision tree as an interpretable model to approximate the behavior of the support vector machine.



The terminal nodes of a surrogate tree that approximates the predictions of a support vector machine trained on the bike rental dataset. The distributions in the nodes show that the surrogate tree predicts a higher number of rented bikes when temperature is above 13 degrees Celsius and when the day was later in the 2 year period (cut point at 435 days).

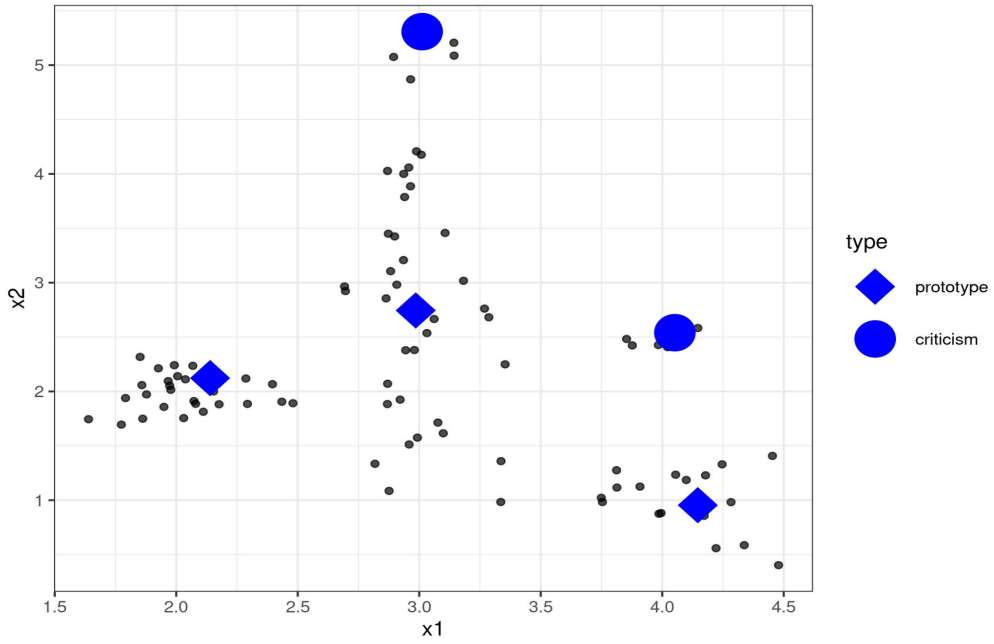
- The surrogate model has a R-squared (variance explained) of 0.77 which means it approximates the underlying black box behavior quite well, but not perfectly. If the fit were perfect, we could throw away the support vector machine and use the tree instead
- **Tools For Implementation** :
    - For Python users, there is the **interpret** package available.
    - https://github.com/interpretml/interpret

Cons:

- We d**raw conclusions about the model and not about the data,** since the surrogate model never sees the real outcome.
- It is not clear **what the best cut-off for R-squared is in order to be confident that the surrogate model is close enough to the black box model.** 80% of variance explained? 50%? 99%?
- We can measure how close the surrogate model is to the black box model. Let us assume we are not very close, but close enough. It could happen that the interpretable model is very close for one subset of the dataset, but widely divergent for another subset. In this case the interpretation for the simple model would not be equally good for all data points.

7. Prototypes and Criticisms

- A prototype is a data instance that is representative of all the data. A criticism is a data instance that is not well represented by the set of prototypes.



- **Maximum mean discrepancy (MMD)**, which measures the discrepancy between two distributions. The selection of prototypes creates a density distribution of prototypes. We want to evaluate whether the prototype distribution differs from the data distribution.

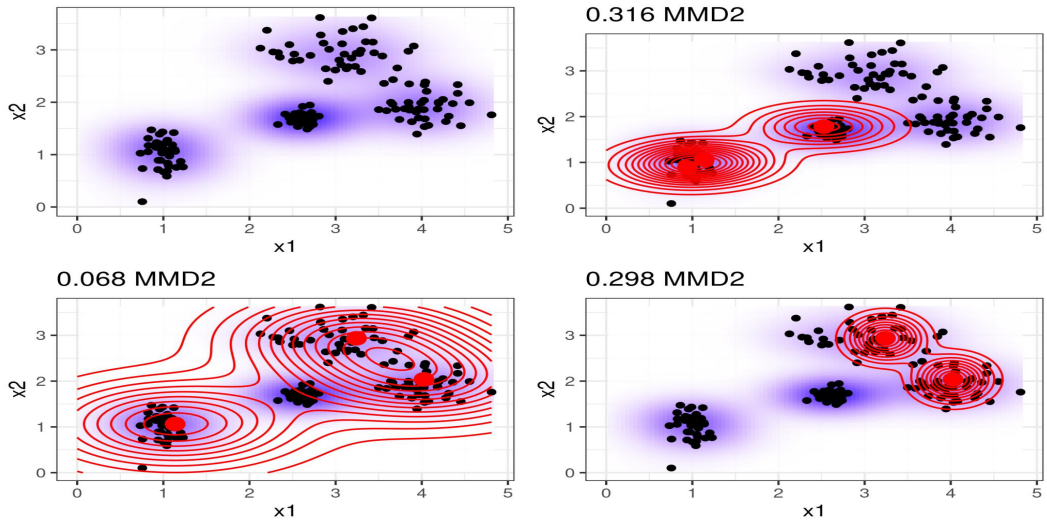
$$MMD^2 = \frac{1}{m^2}\sum_{i,j=1}^{m} k(z_i, z_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(z_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(x_i, x_j)$$

k is a kernel function that measures the similarity of two points, but more about this later. m is the number of prototypes z, and n is the number of data points x in our original dataset. The prototypes z are a selection of data points x. Each point is multidimensional, that is it can have multiple features. The goal of MMD-critic is to minimize MMD2. The closer MMD2 is to zero, the better the distribution of the prototypes fits the data. The key to bringing MMD2 down to zero is the term in the middle, which calculates the average proximity between the prototypes and all other data points (multiplied by 2). If this term adds up to the first term (the average proximity of the prototypes to each other) plus the last term (the average proximity of the data points to each other), then the prototypes explain the data perfectly. Try out what would happen to the formula if you used all n data points as prototypes.

- **The MMD-critic procedure on a high-level can be summarized briefly:**

1. Select the number of prototypes and criticisms you want to find.
2. Find prototypes with greedy search. Prototypes are selected so that the distribution of the prototypes is close to the data distribution.
3. Find criticisms with greedy search. Points are selected as criticisms where the distribution of prototypes differs from the distribution of the data.

- **Example :**
- The following graphic illustrates the MMD2 measure. The first plot shows the data points with two features, whereby the estimation of the data density is displayed with a shaded background. Each of the other plots shows different selections of prototypes, along with the MMD2 measure in the plot titles. **The prototypes are the large dots and their distribution is shown as contour lines**. The selection of the prototypes that best covers the data in these scenarios (bottom left) has the lowest discrepancy value.
-



- ## Concept of witness function

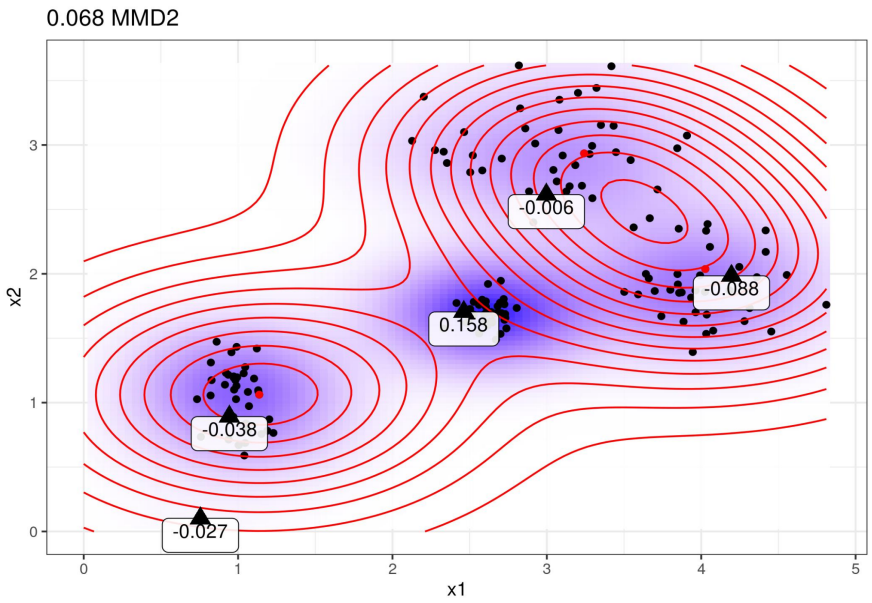
We combine the MMD2 measure, the kernel and greedy search in an algorithm for finding prototypes:

- Start with an empty list of prototypes.
- While the number of prototypes is below the chosen number m:
  - For each point in the dataset, check how much MMD2 is reduced when the point is added to the list of prototypes. Add the data point that minimizes the MMD2 to the list.
- Return the list of prototypes.

The remaining ingredient for finding criticisms is the witness function, which tells us how much two density estimates differ at a particular point. It can be estimated using:

$$witness(x) = \frac{1}{n}\sum_{i=1}^{n} k(x, x_i) - \frac{1}{m}\sum_{j=1}^{m} k(x, z_j)$$

- For two datasets (with the same features), the witness function gives you the means of evaluating in which empirical distribution the point x fits better.
- To find criticisms, we look for extreme values of the witness function in both negative and positive directions. The first term in the witness function is the average proximity between point x and the data, and, respectively, the second term is the average proximity between point x and the prototypes.
- If the witness function for a point x is close to zero, the density function of the data and the prototypes are close together, which means that the distribution of prototypes resembles the distribution of the data at point x.
- A negative witness function at point x means that the prototype distribution overestimates the data distribution (for example if we select a prototype but there are only few data points nearby); a positive witness function at point x means that the prototype distribution underestimates the data distribution (for example if there are many data points around x but we have not selected any prototypes nearby).
- **Example:**
- Let us reuse the prototypes from the plot beforehand with the lowest MMD2 and display the witness function for a few manually selected points. **The labels in the following plot show the value of the witness function for various points marked as triangles**. **Only the point in the middle has a high absolute value and is therefore a good candidate for a criticism**.



- The witness function allows us to explicitly search for data instances that are not well represented by the prototypes. Criticisms are points with high absolute value in the witness function. Like prototypes, criticisms are also found through greedy search.
- But instead of reducing the overall MMD2, we are looking for points that maximize a cost function that includes the witness function and a regularizer term. The additional term in the optimization function enforces diversity in the points, which is needed so that the points come from different clusters.
- **How can MMD-critic be used for interpretable machine learning?**
- MMD-critic can add interpretability in three ways: By helping to better understand the data distribution; by building an interpretable model; by making a black box model interpretable.
- If you apply MMD-critic to your data to find prototypes and criticisms, it will improve your understanding of the data, especially if you have a complex data distribution with edge cases. But with MMD-critic you can achieve more!

For example, you can create an interpretable prediction model: a so-called “nearest prototype model”. The prediction function is defined as:

$$\hat{f}(x) = argmax_{i \in S} k(x, x_i)$$

which means that we select the prototype i from the set of prototypes S that is closest to the new data point, in the sense that it yields the highest value of the kernel function. The prototype itself is returned as an explanation for the prediction. This procedure has three tuning parameters: The type of kernel, the kernel scaling parameter and the number of prototypes. All parameters can be optimized within a cross validation loop. The criticisms are not used in this approach.

As a third option, we can use MMD-critic to make any machine learning model globally explainable by examining prototypes and criticisms along with their model predictions. The procedure is as follows:

1. Find prototypes and criticisms with MMD-critic.
2. Train a machine learning model as usual.
3. Predict outcomes for the prototypes and criticisms with the machine learning model.
4. Analyse the predictions: In which cases was the algorithm wrong? Now you have a number of examples that represent the data well and help you to find the weaknesses of the machine learning model.

- **Remember when Google’s image classifier identified black people as gorillas?**

Perhaps they should have used the procedure described here before deploying their image recognition model. It is not enough just to check the performance of the model, because if it were 99% correct, this issue could still be in the 1%. And labels can also be wrong! Going through all the training data and performing a sanity check if the prediction is problematic might have revealed the problem, but would be infeasible. **But the selection of – say a few thousand – prototypes and criticisms is feasible and could have revealed a problem with the data: It might have shown that there is a lack of images of people with dark skin, which indicates a problem with the diversity in the dataset. Or it could have shown one or more images of a person with dark skin as a prototype or (probably) as a criticism with the notorious “gorilla” classification**. I do not promise that MMD-critic would certainly intercept these kind of mistakes, but it is a good sanity check.

Cons:

- Suppose you choose a too low number of prototypes to cover the data distribution. The criticisms would end up in the areas that are not that well explained. But if you were to add more prototypes they would also end up in the

same areas. Any interpretation has to take into account that **criticisms strongly depend on the existing prototypes and the (arbitrary) cut-off value for the number of prototypes.**

- **How many prototypes and criticisms do we actually need? The more the better? The less the better?** One solution is to select the number of prototypes and criticisms by measuring how much time humans have for the task of looking at the images, which depends on the particular application. **One solution could be a screeplot showing the number of prototypes on the x-axis and the MMD2 measure on the y-axis. We would choose the number of prototypes where the MMD2 curve flattens.**
- **How do we select a kernel and its scaling parameter?** Again, when we use MMD-critic as a nearest prototype classifier, we can tune the kernel parameters. For the unsupervised use cases of MMD-critic, however, it is unclear. (Maybe I am a bit harsh here, since all unsupervised methods have this problem.)
- It **takes all the features as input, disregarding the fact that some features might not be relevant for predicting the outcome of interest.** One solution is to use only relevant features, for example image embeddings instead of raw pixels. This works as long as we have a way to project the original instance onto a representation that contains only relevant information.
- There is some code available, but it is not yet implemented as nicely packaged and documented software.

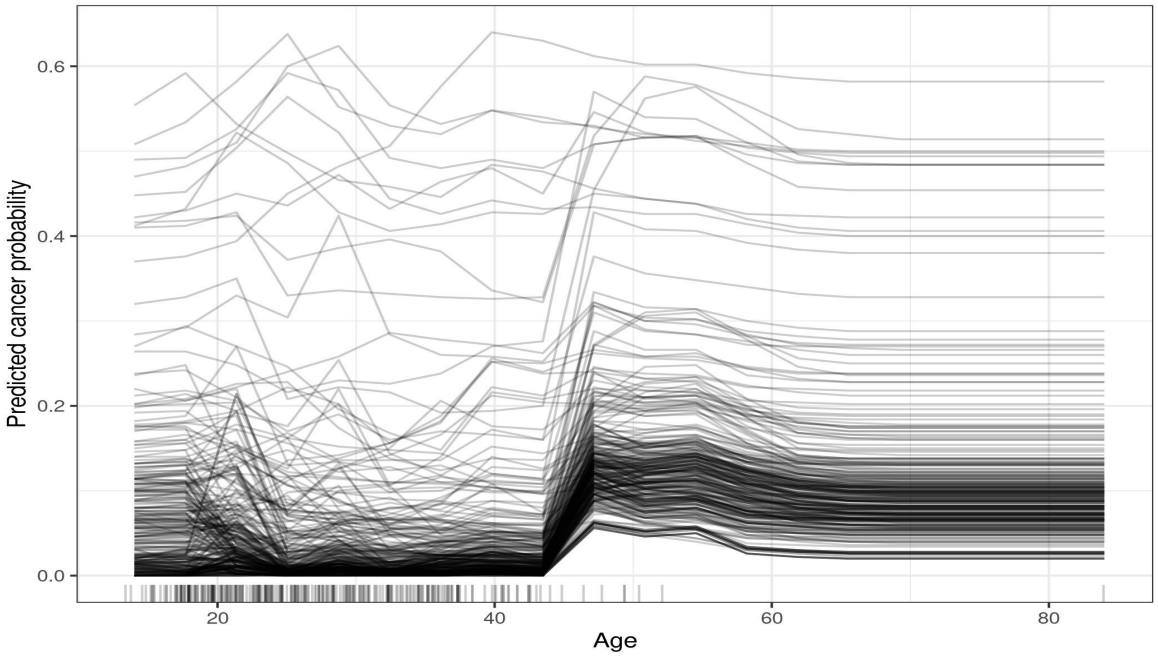
Code

- https://christophm.github.io/interpretable-ml-book/proto.html

Local Agnostic Models:

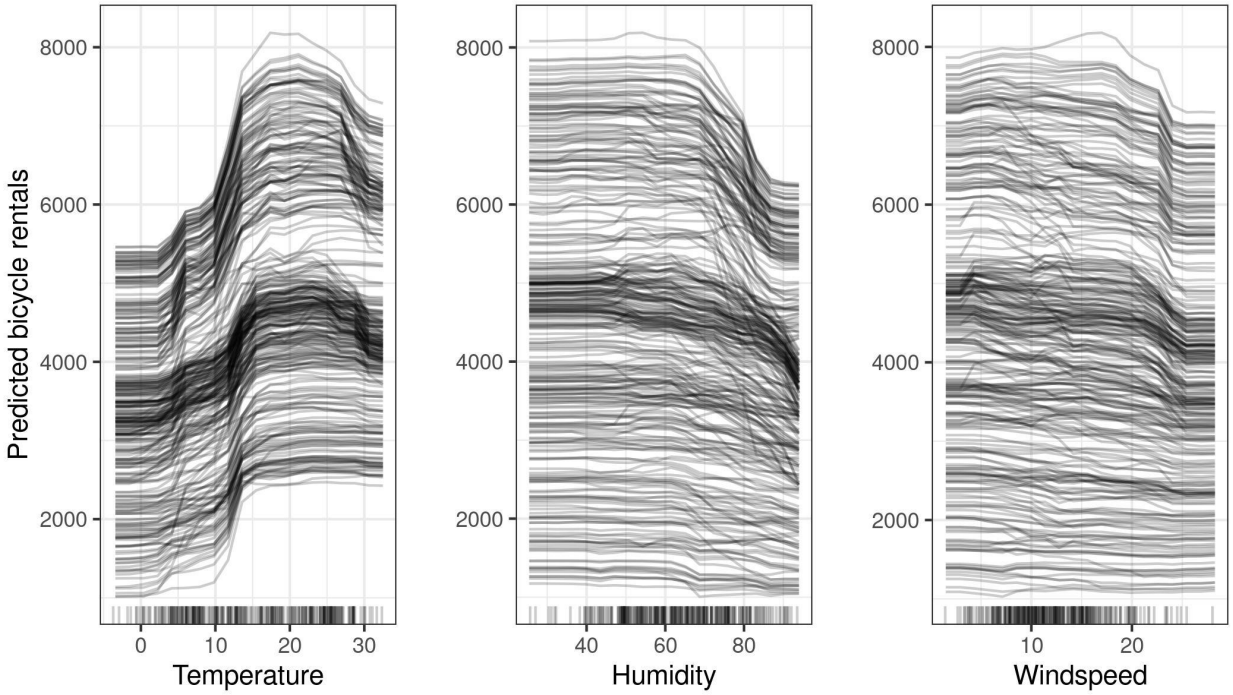
1. Individual Conditional Expectation (ICE)

- Individual Conditional Expectation (ICE) plots **display one line per instance that shows how the instance's prediction changes when a feature changes**.
- **Examples**
- **Classification** Let's go back to the cervical cancer dataset and see how the prediction of each instance is associated with the feature "Age". We will analyze a random forest that predicts the probability of cancer for a woman given risk factors. The ICE plot reveals that for most women the age effect follows the average pattern of an increase at age 50, but there are some exceptions: For the few women that have a high predicted probability at a young age, the predicted cancer probability does not change much with age.



ICE plot of cervical cancer probability by age. Each line represents one woman. For most women there is an increase in predicted cancer probability with increasing age. For some women with a predicted cancer probability above 0.4, the prediction does not change much at higher age.

- **Regression:**The next figure shows ICE plots for the bike rental prediction. The underlying prediction model is a random forest.



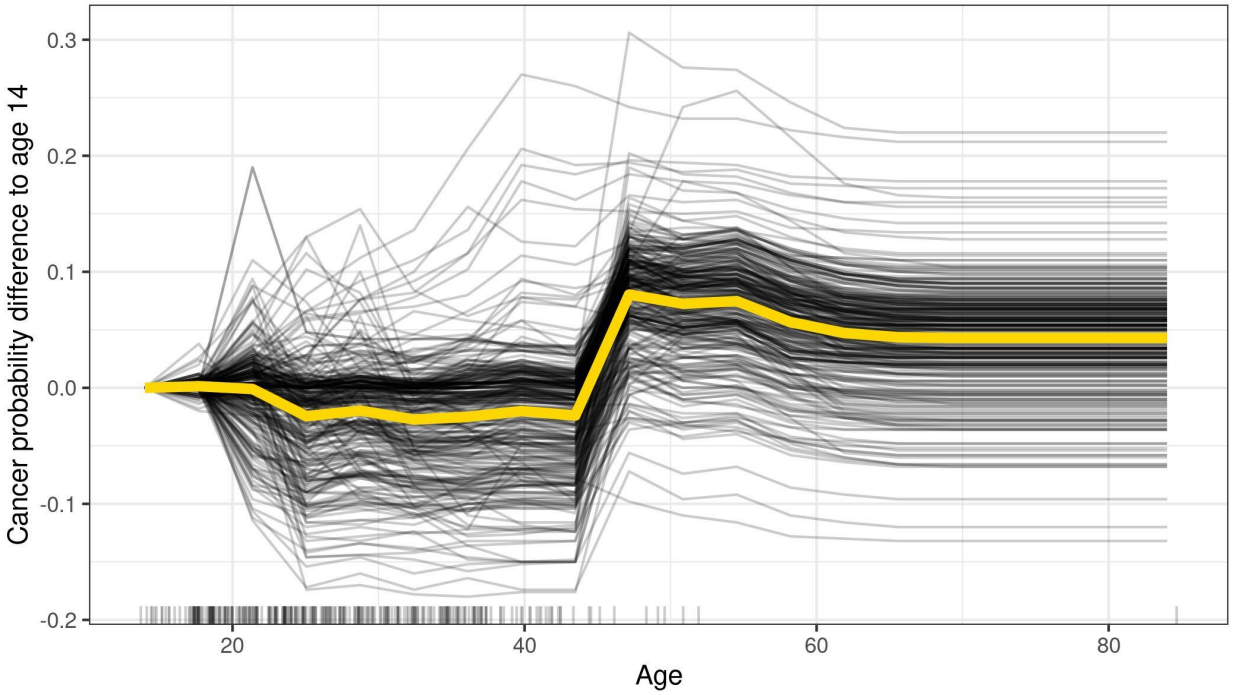
ICE plots of predicted bicycle rentals by weather conditions. The same effects can be observed as in the partial dependence plots.

- All curves seem to follow the same course, so there are no obvious interactions. That means that the PDP is already a good summary of the relationships between the displayed features and the predicted number of bicycles
- **Centered ICE Plot**
- There is a problem with ICE plots: Sometimes it can be hard to tell whether the ICE curves differ between individuals because they start at different predictions. A simple solution is to center the curves at a certain point in the feature and display only the difference in the prediction to this point. The resulting plot is called centered ICE plot (c-ICE). Anchoring the curves at the lower end of the feature is a good choice. The new curves are defined as:

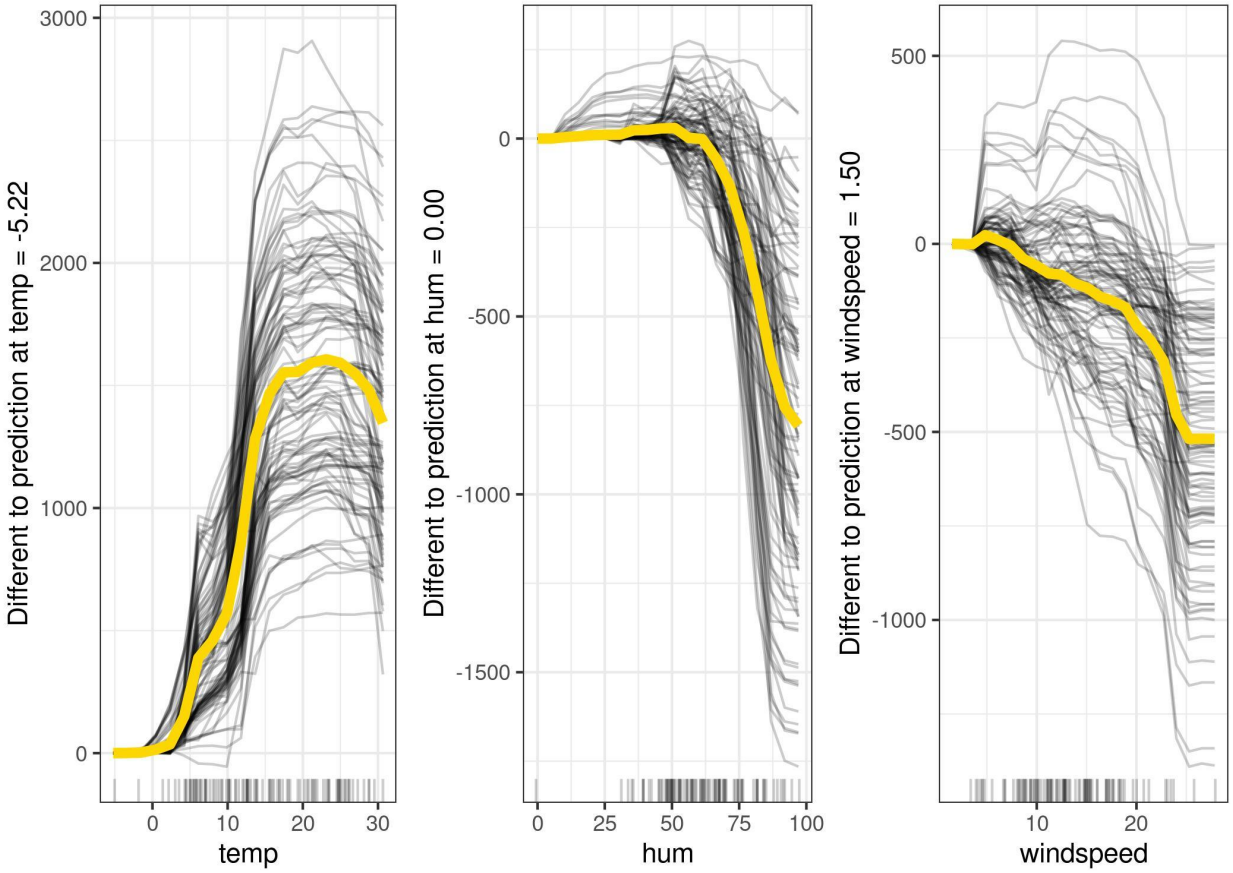
$$\hat{f}_{cent}^{(i)} = \hat{f}^{(i)} - \mathbf{1}\hat{f}(x^a, x_C^{(i)})$$

where 1 is a vector of 1's with the appropriate number of dimensions (usually one or two), f^ is the fitted model and x_a is the anchor point.

- For example, take the cervical cancer ICE plot for age and center the lines on the youngest observed age:



- FIGURE 9.3: Centered ICE plot for predicted cancer probability by age. Lines are fixed to 0 at age 14. Compared to age 14, the predictions for most women remain unchanged until the age of 45 where the predicted probability increases.
- The centered ICE plots make it easier to compare the curves of individual instances. This can be useful if we do not want to see the absolute change of a predicted value, but the difference in the prediction compared to a fixed point of the feature range.
- Let's have a look at centered ICE plots for the bicycle rental prediction:



- FIGURE 9.4: Centered ICE plots of predicted number of bikes by weather condition. The lines show the difference in prediction compared to the prediction with the respective feature value at its observed minimum.

Disadvantages

- ICE curves **can only display one feature** meaningfully, because two features would require the drawing of several overlaying surfaces and you would not see anything in the plot.
- ICE curves suffer from the same problem as PDPs: If the feature of interest is correlated with the other features, then **some points in the lines might be invalid data points** according to the joint feature distribution.
- If many ICE curves are drawn, the **plot can become overcrowded** and you will not see anything. The solution: Either add some transparency to the lines or draw only a sample of the lines.
- In ICE plots it might not be easy to **see the average**. This has a simple solution: Combine individual conditional expectation curves with the partial dependence plot.

References:

1.https://towardsdatascience.com/how-to-explain-and-affect-individual-decisions-with-ice-curves-1-2-f39fd751546f
2. ICE plots are implemented in the R packages `iml` (used for these examples), `ICEbox`[49], and `pdp`. Another R package that does something very similar to ICE is `condvis`. In Python, partial dependence plots are built into scikit-learn starting with version 0.24.0.

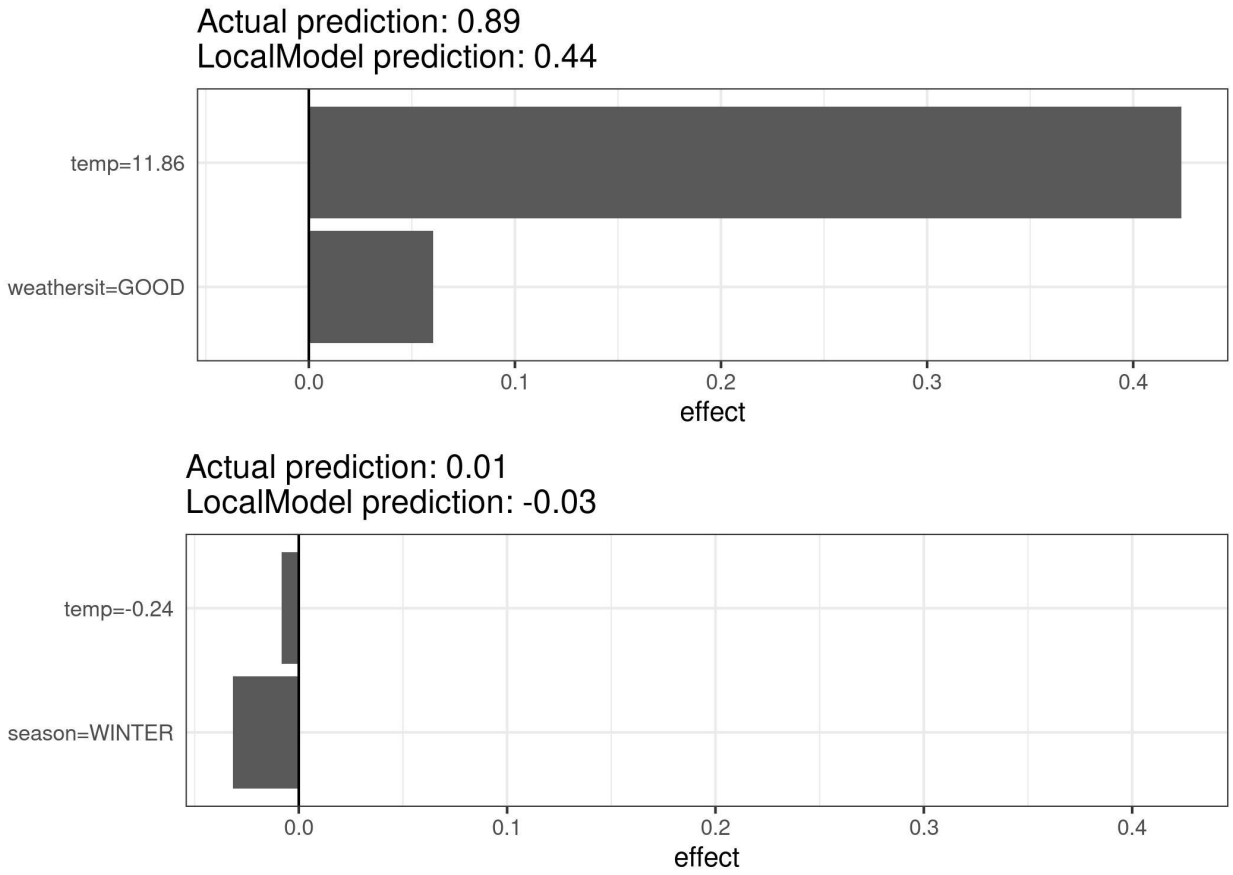
2. Local Surrogate (LIME)

The recipe for training local surrogate models:

- **Select your instance of interest for which you want to have an explanation of its black box prediction.**
- **Perturb your dataset and get the black box predictions for these new points.**
- **Weight the new samples according to their proximity to the instance of interest.**
- **Train a weighted, interpretable model on the dataset with the variations.**
- **Explain the prediction by interpreting the local model.**

Example for Tabular Data

- Let us look at a concrete example. We go back to the bike rental data and turn the prediction problem into a classification: After taking into account the trend that bicycle rental has become more popular over time, we want to know on a certain day whether the number of bicycles rented will be above or below the trend line. You can also interpret “above” as being above the average number of bicycles, but adjusted for the trend.
- First we train a random forest with 100 trees on the classification task. On what day will the number of rental bikes be above the trend-free average, based on weather and calendar information?
- The explanations are created with 2 features. The results of the sparse local linear models trained for two instances with different predicted classes:



- LIME explanations for two instances of the bike rental dataset. Warmer temperature and good weather situation have a positive effect on the prediction. The x-axis shows the feature effect: The weight times the actual feature value.
- From the figure it becomes clear that it is easier to interpret categorical features than numerical features. One solution is to categorize the numerical features into bins.

Example for text

In this example we classify YouTube comments as spam or normal.

Let us look at the two comments of this dataset and the corresponding classes (1 for spam, 0 for normal comment):

	CONTENT	CLASS
267	PSY is a good guy	0
173	For Christmas Song visit my channel! ;)	1

The next step is to create some variations of the datasets used in a local model. For example, some variations of one of the comments:

For	Christmas	Song	visit	my	channel!	;)	prob	weight
1	0	1	1	0	0	1	0.17	0.57
0	1	1	1	1	0	1	0.17	0.71
1	0	0	1	1	1	1	0.99	0.71
1	0	1	1	1	1	1	0.99	0.86
0	1	1	1	0	0	1	0.17	0.57

Each column corresponds to one word in the sentence. Each row is a variation, 1 means that the word is

- Each column corresponds to one word in the sentence. Each row is a variation, 1 means that the word is part of this variation and 0 means that the word has been removed. The corresponding sentence for one of the variations is "`Christmas Song visit my ;)`". The "prob" column shows the predicted probability of spam for each of the sentence variations. The "weight" column shows the proximity of the variation to the original sentence, calculated as 1 minus the proportion of words that were removed, for example if 1 out of 7 words was removed, the proximity is 1 - 1/7 = 0.86.

Here are the two sentences (one spam, one no spam) with their estimated local weights found by the LIME algorithm:

case	label_prob	feature	feature_weight
1	0.1701170	PSY	0.000000
1	0.1701170	guy	0.000000
1	0.1701170	good	0.000000
2	0.9939024	channel!	6.180747
2	0.9939024	;)	0.000000
2	0.9939024	visit	0.000000

The word "channel" indicates a high probability of spam. For the non-spam comment no non-zero weight was estimated, because no matter which word is removed, the predicted class remains the same.

Example for Images:

- In this example we look at a classification made by the Inception V3 neural network. The image used shows some bread I baked which are in a bowl. Since we can have several predicted labels per image (sorted by probability), we can explain the top labels. The top prediction is “Bagel” with a probability of 77%, followed by “Strawberry” with a probability of 4%. The following images show for “Bagel” and “Strawberry” the LIME explanations. The explanations can be displayed directly on the image samples. Green means that this part of the image increases the probability for the label and red means a decrease.

: Left: Image of a bowl of bread. Middle and right: LIME explanations for the top 2 classes (bagel, strawberry) for image classification made by Google’s Inception V3 neural network.

- The prediction and explanation for “Bagel” are very reasonable, even if the prediction is wrong – these are clearly no bagels since the hole in the middle is missing.

Cons:

- **The correct definition of the neighborhood is a very big,** unsolved problem when using LIME with tabular data. In my opinion it is the biggest problem with LIME and the reason why I would recommend to use LIME only with great care. For each application you have to try different kernel settings and see for yourself if the explanations make sense. **Unfortunately, this is the best advice I can give to find good kernel widths.Sampling could be improved in the current implementation of LIME.** Data points are sampled from a Gaussian distribution, ignoring the correlation between features. This can lead to unlikely data points which can then be used to learn local explanation models.

- Another really big problem is the instability of the explanations. **In an article [51] the authors showed that the explanations of two very close points varied greatly in a simulated setting. Also, in my experience, if you repeat the sampling process, then the explanations that come out can be different.** Instability means that it is difficult to trust the explanations, and you should be very critical.
- **LIME explanations can be manipulated by the data scientist to hide biases** [52]. The possibility of manipulation makes it more difficult to trust explanations generated with LIME.
- Conclusion: Local surrogate models, with LIME as a concrete implementation, are very promising. But the method is still in the development phase and many problems need to be solved before it can be safely applied.

3.Counterfactual Explanations

- A counterfactual explanation describes a causal situation in the form: "If X had not occurred, Y would not have occurred". For example: "If I hadn't taken a sip of this hot coffee, I wouldn't have burned my tongue". Event Y is that I burned my tongue; cause X is that I had a hot coffee. Thinking in counterfactuals requires imagining a hypothetical reality that contradicts the observed facts (for example, a world in which I have not drunk the hot coffee), hence the name "counterfactual". The ability to think in counterfactuals makes us humans so smart compared to other animals.

    1. Select an instance x to be explained, the desired outcome y', a tolerance ϵ and a (low) initial value for λ.
    2. Sample a random instance as initial counterfactual.
    3. Optimize the loss with the initially sampled counterfactual as starting point.
    4. While $|\hat{f}(x') - y'| > \epsilon$:
        - Increase λ.
        - Optimize the loss with the current counterfactual as starting point.
        - Return the counterfactual that minimizes the loss.
    5. Repeat steps 2-4 and return the list of counterfactuals or the one that minimizes the loss.
-

- ## Example
- The following example is based on the credit dataset example in Dandl et al. (2020). The German Credit Risk dataset can be found on the machine learning challenges platform kaggle.com.
- The authors trained a support vector machine (with radial basis kernel) to predict the probability that a customer has a good credit risk. The corresponding dataset has 522 complete observations and nine features containing credit and customer information.

- The goal is to find counterfactual explanations for a customer with the following feature values:

age	sex	job	housing	savings	amount	duration	purpose
58	f	unskilled	free	little	6143	48	car

The SVM predicts that the woman has a good credit risk with a probability of 24.2 %. The counterfactuals should answer how the input features need to be changed to get a predicted probability larger than 50 %?

The following table shows the ten best counterfactuals:

age	sex	job	amount	duration	o_2	o_3	o_4	$\hat{f}(x')$
		skilled		-20	0.108	2	0.036	0.501
		skilled		-24	0.114	2	0.029	0.525
		skilled		-22	0.111	2	0.033	0.513
-6		skilled		-24	0.126	3	0.018	0.505
-3		skilled		-24	0.120	3	0.024	0.515
-1		skilled		-24	0.116	3	0.027	0.522
-3	m			-24	0.195	3	0.012	0.501
-6	m			-25	0.202	3	0.011	0.501
-30	m	skilled		-24	0.285	4	0.005	0.590
-4	m		-1254	-24	0.204	4	0.002	0.506

- The first five columns contain the proposed feature changes (only altered features are displayed), the next three columns show the objective values (o1 equals 0 in all cases) and the last column displays the predicted probability.
- All counterfactuals have predicted probabilities greater than 50 % and do not dominate each other. Nondominated means that none of the counterfactuals has smaller values in all objectives than the other counterfactuals. We can think of our counterfactuals as a set of trade-off solutions.
- They all suggest a reduction of the duration from 48 months to minimum 23 months, some of them propose that the woman should become skilled instead of unskilled. Some counterfactuals even suggest changing the gender from female to male which shows a gender bias of the model. This change is always accompanied by a reduction in age between one and 30 years. We can also see that, although some counterfactuals suggest changes to four features, these counterfactuals are the ones that are closest to the training data.

Disadvantages

- **For each instance you will usually find multiple counterfactual explanations (Rashomon effect)**. This is inconvenient - most people prefer simple explanations over the complexity of the real world. It is also a practical challenge.
- Let us say we generated 23 counterfactual explanations for one instance. Are we reporting them all? Only the best? What if they are all relatively “good”, but very different?
- These questions must be answered anew for each project. It can also be advantageous to have multiple counterfactual explanations, because humans then can select the ones that correspond to their previous knowledge.

Sources:

- https://github.com/dandls/moc/tree/master/counterfactuals
- https://docs.seldon.io/projects/alibi/en/stable/methods/CF.html
- https://docs.seldon.io/projects/alibi/en/stable/methods/CFProto.html
- https://github.com/amirhk/mace
- https://github.com/interpretml/DiCE

4. Scoped Rules (Anchors)

The anchors method explains individual predictions of any black box classification model by finding a decision rule that “anchors” the prediction sufficiently

Anchors utilizes **reinforcement learning** techniques in combination with a graph search algorithm to reduce the number of model calls (and hence the required runtime) to a minimum while still being able to recover from local optima.

Like its predecessor, the anchors approach deploys a *perturbation-based* strategy to generate *local* explanations for predictions of black box machine learning models.

However, instead of surrogate models used by LIME, the resulting explanations are expressed as easy-to-understand *IF-THEN* rules, called *anchors*.

These rules are reusable, since they are *scoped*: anchors include the notion of coverage, stating precisely to which other, possibly unseen, instances they apply.

Finding anchors involves an exploration or multi-armed bandit problem, which originates in the discipline of reinforcement learning. To this end, neighbors, or perturbations, are created and evaluated for every instance that is being explained.

Doing so allows the approach to disregard the black box's structure and its internal parameters so that these can remain both unobserved and unaltered. Thus, the algorithm is *model-agnostic*, meaning it can be applied to **any** class of model.

As mentioned before, the algorithm's results or explanations come in the form of rules, called anchors. The following simple example illustrates such an anchor. For instance, suppose we are given a bivariate black box model that predicts whether or not a passenger survived the Titanic disaster.

Now we would like to know *why* the model predicts for one specific individual that it survives. The anchors algorithm provides a result explanation like the one shown below.

Feature	Value
Age	20
Sex	female
Class	first
Ticket price	300$
More attributes	...
Survived	true

And the corresponding anchors explanation is:

IF `SEX = female` AND `Class = first` THEN PREDICT `Survived = true` WITH PRECISION 97% AND COVERAGE 15%

The example shows how anchors can provide essential insights into a model's prediction and its underlying reasoning. The result shows which attributes were taken into account by the model, which in this case, is the female sex and first class. Humans, being paramount for correctness, can use this rule to validate the model's behavior. The anchor additionally tells us that it applies to 15% of perturbation space's instances. In those cases the explanation is 97% accurate, meaning the displayed predicates are almost exclusively responsible for the predicted outcome.

An anchor A is formally defined as follows:

$$\mathbb{E}_{\mathcal{D}_x(z|A)}[1_{\hat{f}(x)=\hat{f}(z)}] \geq \tau, A(x) = 1$$

Wherein:

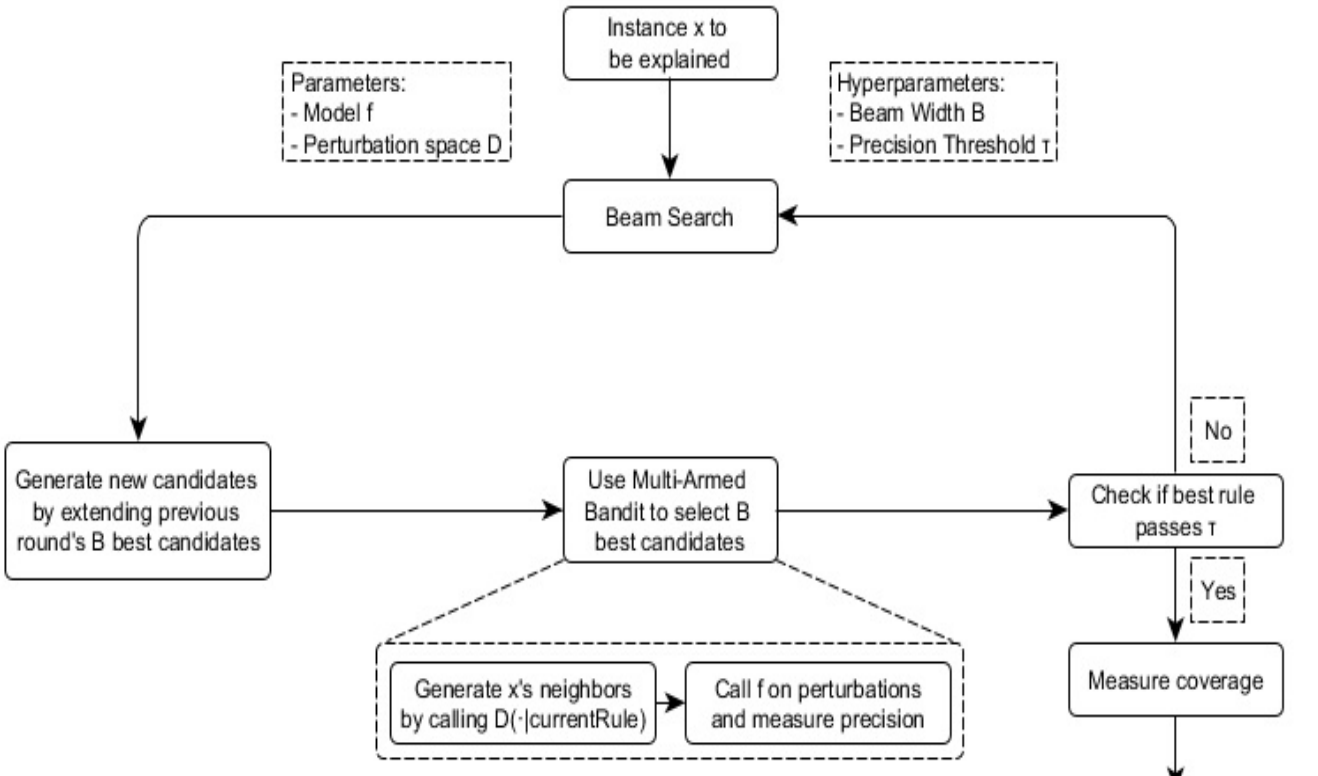
- x represents the instance being explained (e.g. one row in a tabular data set).
- A is a set of predicates, i.e., the resulting rule or anchor, such that $A(x) = 1$ when all feature predicates defined by A correspond to x's feature values.
- f denotes the classification model to be explained (e.g. an artificial neural network model). It can be queried to predict a label for x and its perturbations.
- $D_x(\cdot|A)$ indicates the distribution of neighbors of x, matching A.
- $0 \leq \tau \leq 1$ specifies a precision threshold. Only rules that achieve a local fidelity of at least τ are considered a valid result.

The formal description may be intimidating and can be put in words:

> Given an instance x to be explained, a rule or an anchor A is to be found, such that it applies to x, while the same class as for x gets predicted for a fraction of at least τ of x's neighbors where the same A is applicable. A rule's precision results from evaluating neighbors or perturbations (following $D_x(z|A)$) using the provided machine learning model (denoted by the indicator function $1_{\hat{f}(x)=\hat{f}(z)}$).

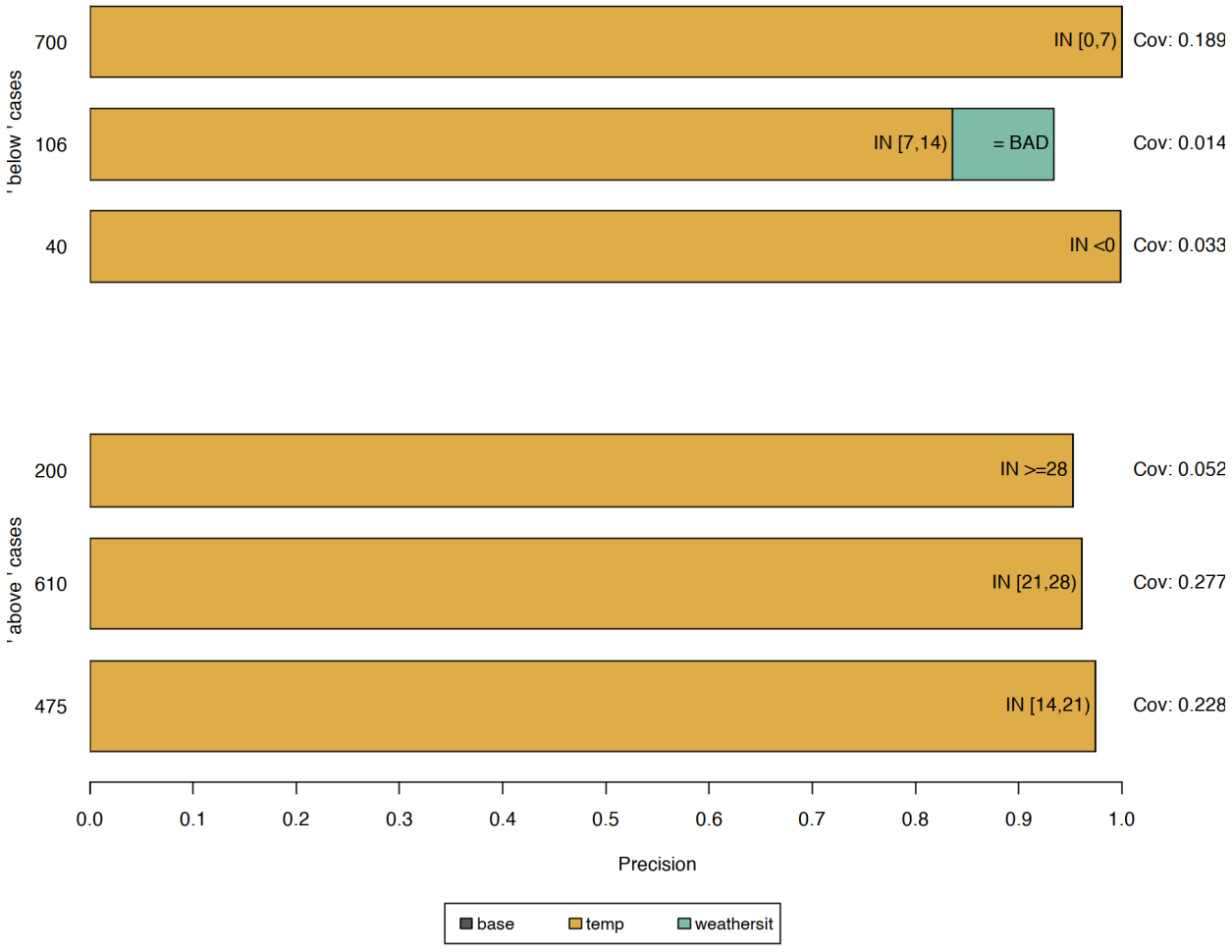
- The anchors approach uses four main components to find explanations.
- **Candidate Generation**: Generates new explanation candidates. In the first round, one candidate per feature of X gets created and fixes the respective value of possible perturbations. In every other round, the best candidates of the previous round are extended by one feature predicate that is not yet contained therein.
- **Best Candidate Identification**: Candidate rules are to be compared in regard to which rule explains x the best. To this end, perturbations that match the currently observed rule are created and evaluated by calling the model. However, these calls need to be minimized as to limit computational overhead. This is why, at the core of this component, there is a pure-exploration Multi-Armed-Bandit (*MAB*; *KL-LUCB*[62], to be precise). MABs are used to efficiently explore and exploit different strategies (called arms in an analogy to slot machines) using sequential selection. In the given setting, each candidate rule is to be seen as an arm that can be pulled. Each time it is pulled, respective neighbors get evaluated, and we thereby obtain more information about the candidate rule's payoff (precision in anchor's case). The precision thus states how well the rule describes the instance to be explained.

- **Candidate Precision Validation**: Takes more samples in case there is no statistical confidence yet that the candidate exceeds the τ threshold.
- **Modified Beam Search**: All of the above components are assembled in a beam search, which is a graph search algorithm and a variant of the breadth-first algorithm. It carries the B best candidates of each round over to the next one (where B is called the *Beam Width*). These B best rules are then used to create new rules. The beam search conducts at most $\text{featureCount}(x)$ rounds, as each feature can only be included in a rule at most once. Thus, at every round i, it generates candidates with exactly i predicates and selects the B best thereof. Therefore, by setting B high, the algorithm is more likely to avoid local optima. In turn, this requires a high number of model calls and thereby increases the computational load.
- These four components are shown in the figure below.



Tabular Data Example

- **Regression Problem**: For bike rental data, we turn the regression into a classification problem and train a random forest as our black box model. It is to classify whether the number of rented bicycles lies above or below the trend line.
- Before creating anchor explanations, one needs to define a perturbation function. An easy way to do so is to use an intuitive default perturbation space for tabular explanation cases which can be built by sampling from, e.g. the training data.
- When perturbing an instance, this default approach maintains the feature's values that are subject to the anchors' predicates, while replacing the non-fixed features with values taken from another randomly sampled instance with a specified probability.
- This process yields new instances that are similar to the explained one, but have adopted some values from other random instances. Thus, they resemble neighbors of the explained instance.



Anchors explaining six instances of the bike rental dataset. Each row represents one explanation or anchor, and each bar depicts the feature predicates contained by it. The x-axis displays a rule's precision, and a bar's thickness corresponds to its coverage. The "base" rule contains no predicates. These anchors show that the model mainly considers the temperature for predictions.

- The results are instinctively interpretable and show for each explained instance, which features are most important for the model's prediction. As the anchors only have a few predicates, they additionally have high coverage and hence apply to other cases. The rules shown above were generated with τ=0.9. Thus, we ask for anchors whose evaluated perturbations faithfully support the label with an accuracy of at least 90%. Also, discretization was used to increase the expressiveness and applicability of numerical features.
- All of the previous rules were generated for instances where the model decides confidently based on a few features. However, other instances are not as distinctly classified by the model as more features are of importance. In such cases, anchors get more specific, comprise more features, and apply to fewer instances.

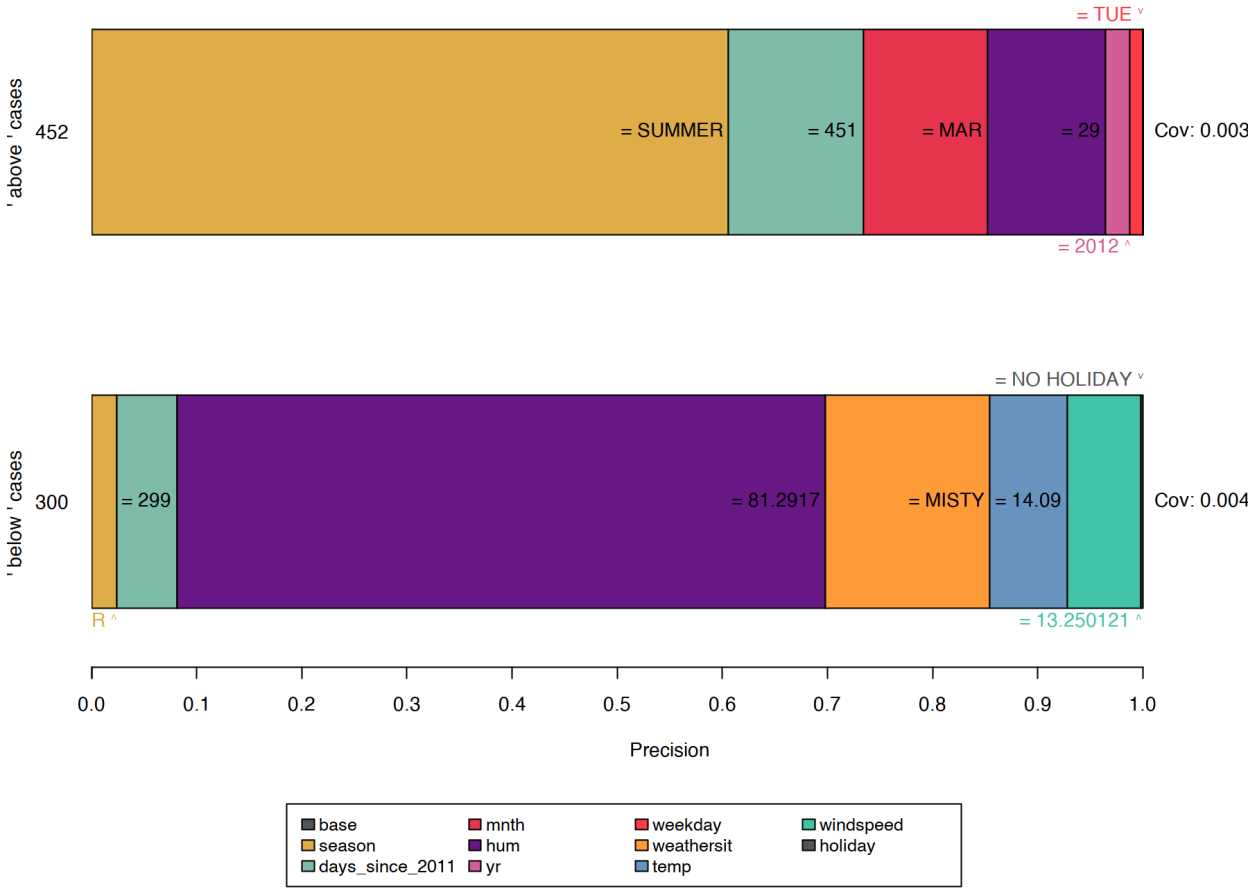
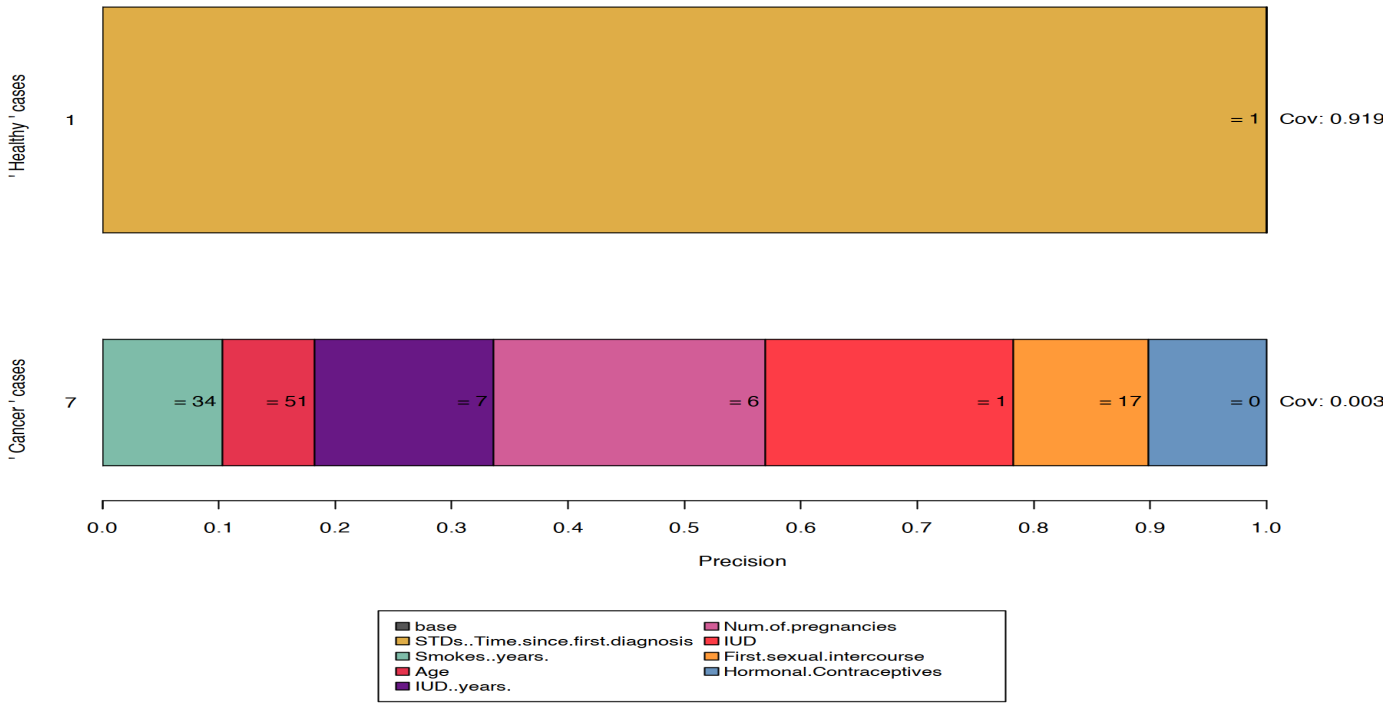


FIGURE 9.14: Explaining instances near decision boundaries leads to specific rules comprising a higher number of feature predicates and lower coverage. Also, the empty rule, i.e. the base feature, gets less important. This can be interpreted as a signal for a decision boundary, as the instance is located in a volatile neighborhood.

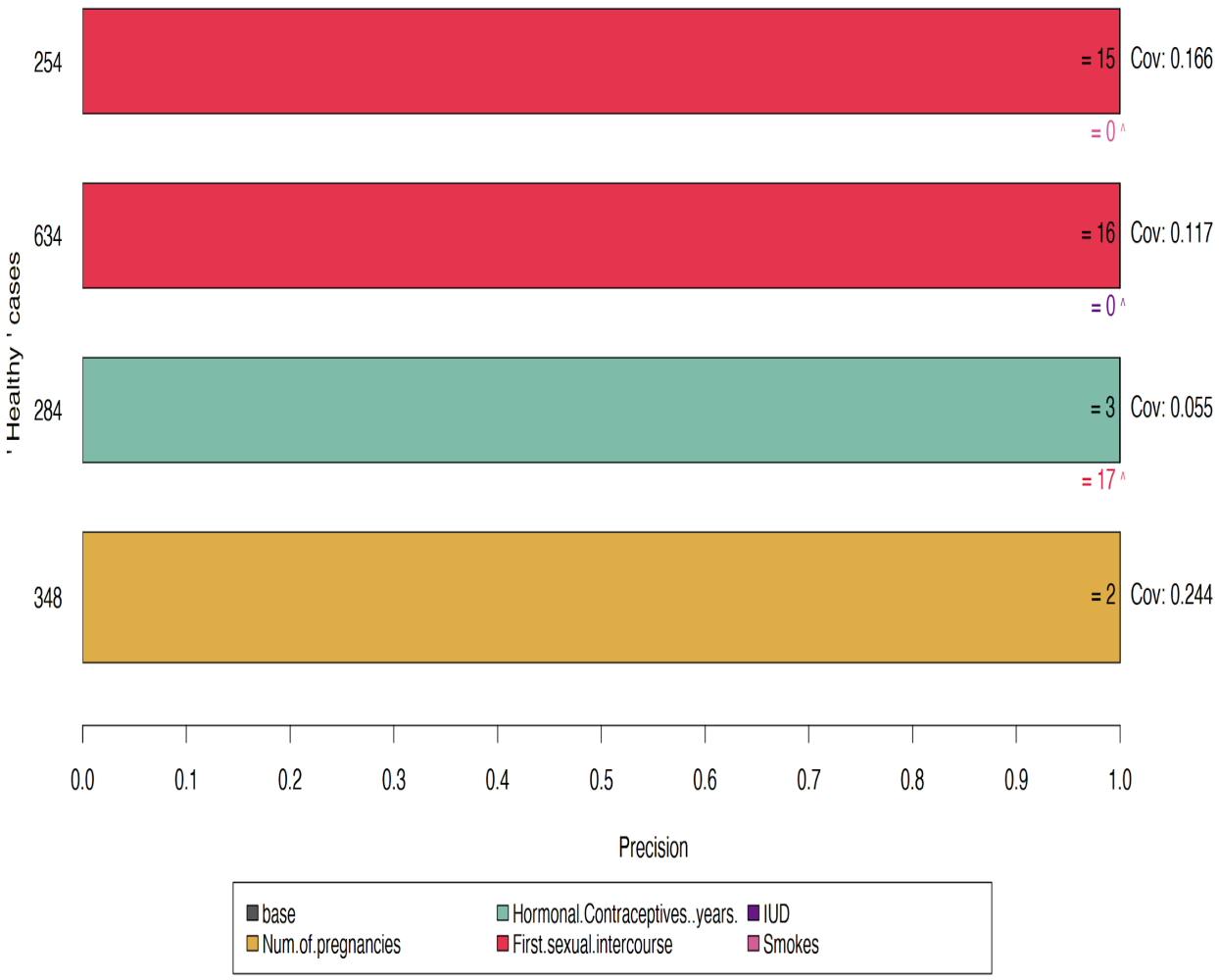
- While choosing the default perturbation space is a comfortable choice to make, it may have a great impact on the algorithm and can thus lead to biased results. For example, if the train set is unbalanced (there is an unequal number of instances of each class), the perturbation space is as well. This condition further affects the rule-finding and the result's precision.

Classification Problem Example

- The cervical cancer data set is an excellent example of this situation. Applying the anchors algorithm leads to one of the following situations:
- Explaining instances labeled *healthy* yields empty rules as all generated neighbors evaluate to *healthy*.
- Explanations for instances labeled *cancer* are overly specific, i.e. comprise many feature predicates, since the perturbation space mostly covers values from *healthy* instances.



- This outcome may be unwanted and can be approached in multiple ways. For example, a custom perturbation space can be defined. This custom perturbation can sample differently, e.g. from an unbalanced data set or a normal distribution.
- This, however, comes with a side-effect: the sampled neighbors are not representative and change the coverage's scope. Alternatively, we could modify the MAB's confidence δ and error parameter values ϵ. This would cause the MAB to draw more samples, ultimately leading to the minority being sampled more often in absolute terms.
- For this example, we use a subset of the cervical cancer set in which the majority of cases are labeled *cancer*. We then have the framework to create a corresponding perturbation space from it. Perturbations are now more likely to lead to varying predictions, and the anchors algorithm can identify important features.
- However, one needs to take the coverage's definition into account: it is only defined within the perturbation space. In the previous examples, we used the train set as the perturbation space's basis. Since we only use a subset here, a high coverage does not necessarily indicate globally high rule importance.



Balancing the data set before constructing anchors shows the model's reasoning for decisions in minority cases.

Disadvantages:

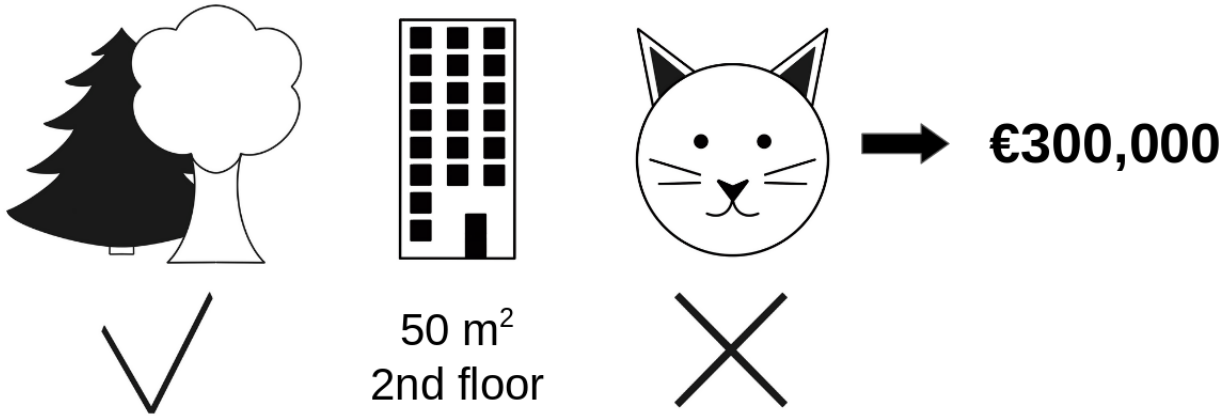
- The algorithm suffers from a **highly configurable** and impactful setup, just like most perturbation-based explainers. Not only do hyperparameters such as the beam width or precision threshold need to be tuned to yield meaningful results, but also does the perturbation function need to be explicitly designed for one domain/use case. Think of how tabular data get perturbed and think of how to apply the same concepts to image data (hint: these cannot be applied). Luckily, default approaches may be used in some domains (e.g. tabular), facilitating an initial explanation setup.

- Also, **many scenarios require discretization** as otherwise results are too specific, have low coverage, and do not contribute to understanding the model. While discretization can help, it may also blur decision boundaries if used carelessly and thus have the exact opposite effect. Since there is no best discretization technique, users need to be aware of the data before deciding on how to discretize data not to obtain poor results.
- Constructing anchors requires **many calls to the ML model**, just like all perturbation-based explainers. While the algorithm deploys MABs to minimize the number of calls, its runtime still very much depends on the model's performance and is therefore highly variable.
- Lastly, the notion of **coverage is undefined in some domains**. For example, there is no obvious or universal definition of how superpixels in one image compare to such in other images.
- **Code: https://github.com/marcotcr/anchor**

5. Shapley Values:

- The interpretation of the Shapley value is: Given the current set of feature values, the contribution of a feature value to the difference between the actual prediction and the mean prediction is the estimated Shapley value.
- **Assume the following scenario:**

You have trained a machine learning model to predict apartment prices. For a certain apartment it predicts €300,000 and you need to explain this prediction. The apartment has an area of 50 m2, is located on the 2nd floor, has a park nearby and cats are banned:

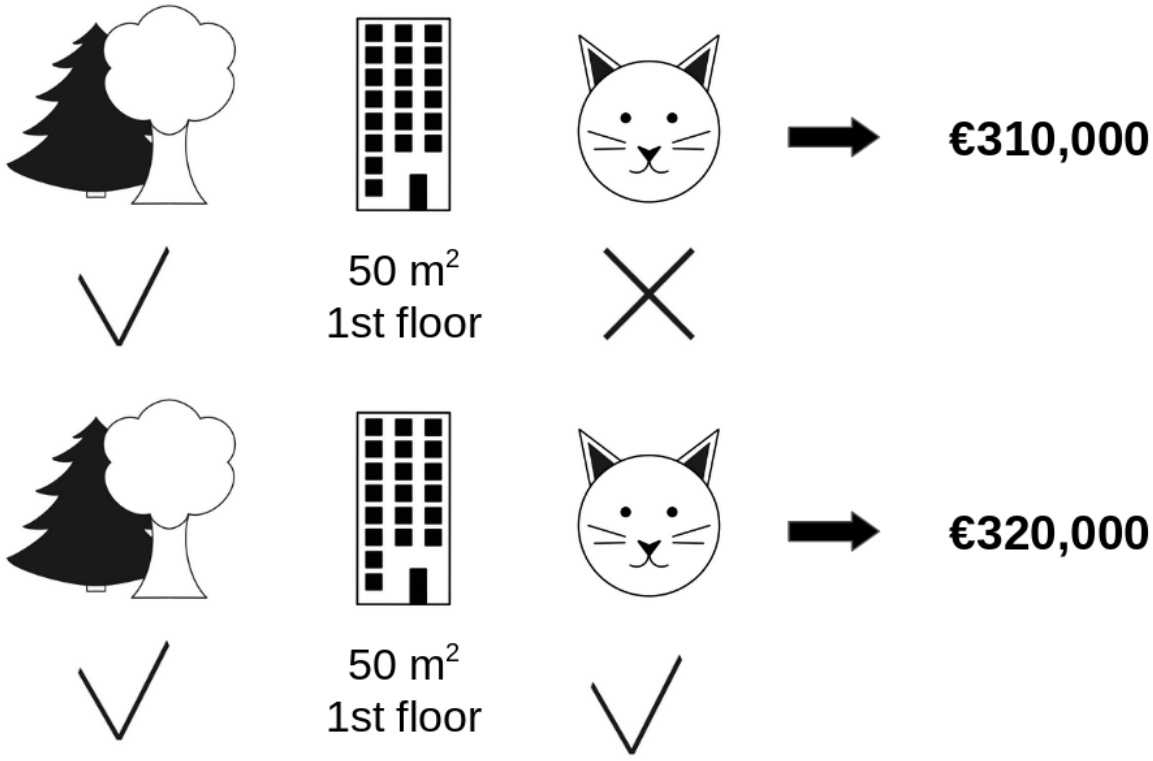


The predicted price for a 50 m2 2nd floor apartment with a nearby park and cat ban is €300,000. Our goal is to explain how each of these feature values contributed to the prediction.

- The average prediction for all apartments is €310,000. How much has each feature value contributed to the prediction compared to the average prediction?
- The answer is simple for linear regression models. The effect of each feature is the weight of the feature times the feature value. This only works because of the linearity of the model. For more complex models, we need a different solution.
- Another solution comes from cooperative game theory: The Shapley value, coined by Shapley (1953)[63], is a method for assigning payouts to players depending on their contribution to the total payout. Players cooperate in a coalition and receive a certain profit from this cooperation.
- Players? Game? Payout? What is the connection to machine learning predictions and interpretability? The “game” is the prediction task for a single instance of the dataset. The “gain” is the actual prediction for this instance minus the average prediction for all instances. The “players” are the feature values of the instance that collaborate to receive the gain (= predict a certain value).
- In our apartment example, the feature values `park-nearby`, `cat-banned`, `area-50` and `floor-2nd` worked together to achieve the prediction of €300,000. Our goal is to explain the difference between the actual prediction (€300,000) and the average prediction (€310,000): a difference of -€10,000.
- The answer could be: The `park-nearby` contributed €30,000; `area-50` contributed €10,000; `floor-2nd` contributed €0; `cat-banned` contributed -€50,000. The contributions add up to -€10,000, the final prediction minus the average predicted apartment price.

How do we calculate the Shapley value for one feature?

- The Shapley value is the average marginal contribution of a feature value across all possible coalitions.
- In the following figure we evaluate the contribution of the `cat-banned` feature value when it is added to a coalition of `park-nearby` and `area-50`. We simulate that only `park-nearby`, `cat-banned` and `area-50` are in a coalition by randomly drawing another apartment from the data and using its value for the floor feature. The value `floor-2nd` was replaced by the randomly drawn `floor-1st`.
- Then we predict the price of the apartment with this combination (€310,000).
- In a second step, we remove `cat-banned` from the coalition by replacing it with a random value of the cat allowed/banned feature from the randomly drawn apartment.
- In the example it was `cat-allowed`, but it could have been `cat-banned` again.
- We predict the apartment price for the coalition of `park-nearby` and `area-50` (€320,000).
- The contribution of `cat-banned` was €310,000 - €320,000 = -€10,000. This estimate depends on the values of the randomly drawn apartment that served as a “donor” for the cat and floor feature values. We will get better estimates if we repeat this sampling step and average the contributions.



One sample repetition to estimate the contribution of `cat-banned` to the prediction when added to the coalition of `park-nearby` and `area-50`.

- We repeat this computation for all possible coalitions. The Shapley value is the average of all the marginal contributions to all possible coalitions. The computation time increases exponentially with the number of features. One solution to keep the computation time manageable is to compute contributions for only a few samples of the possible coalitions.
- The following figure shows all coalitions of feature values that are needed to determine the Shapley value for `cat-banned`. The first row shows the coalition without any feature values. The second, third and fourth rows show different coalitions with increasing coalition size, separated by "|". All in all, the following coalitions are possible:

`No feature values, park-nearby, area-50, floor-2nd, park-nearby`+`area-50, park-nearby`+`floor-2nd, area-50`+`floor-2nd, park-nearby`+`area-50`+`floor-2nd`.

- For each of these coalitions we compute the predicted apartment price with and without the feature value `cat-banned` and take the difference to get the marginal contribution.

- The Shapley value is the (weighted) average of marginal contributions. We replace the feature values of features that are not in a coalition with random feature values from the apartment dataset to get a prediction from the machine learning model.

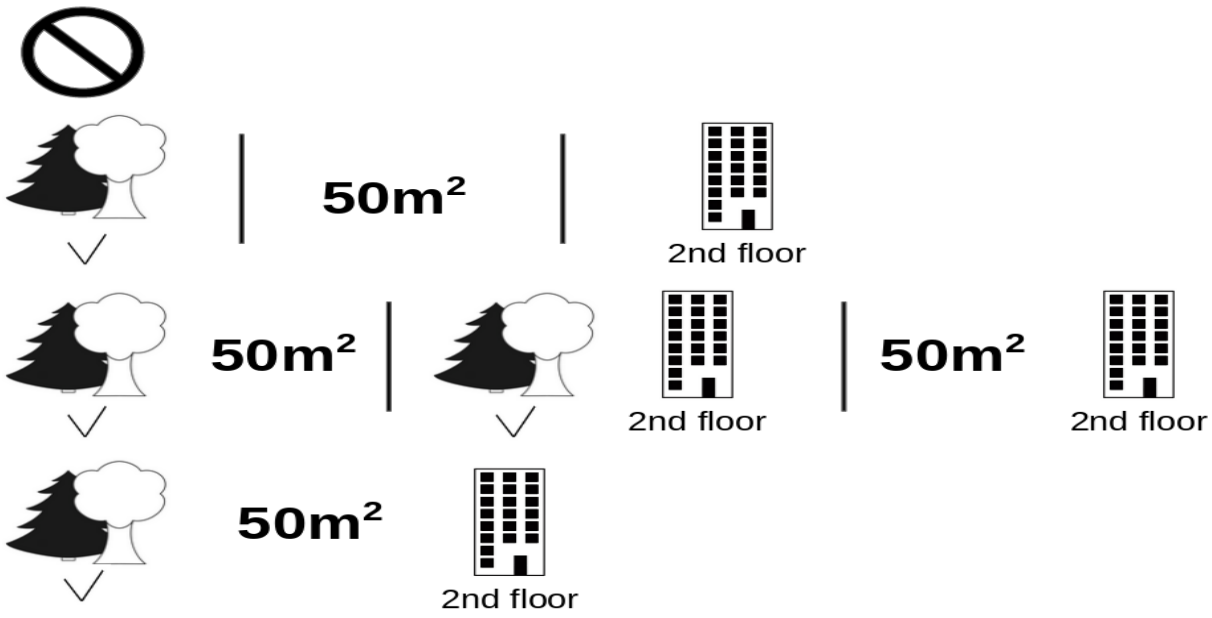
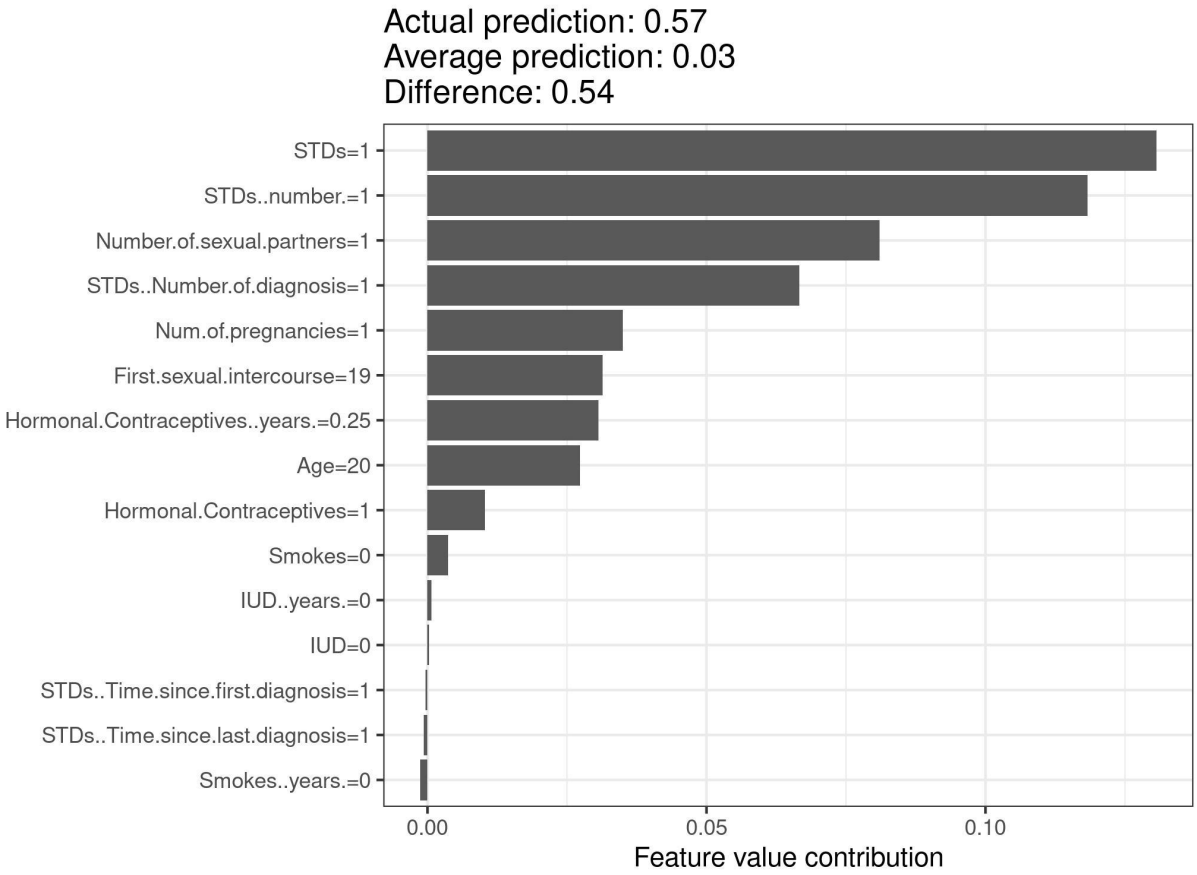


FIGURE: All 8 coalitions needed for computing the exact Shapley value of the `cat-banned` feature value.

- If we estimate the Shapley values for all feature values, we get the complete distribution of the prediction (minus the average) among the feature values.

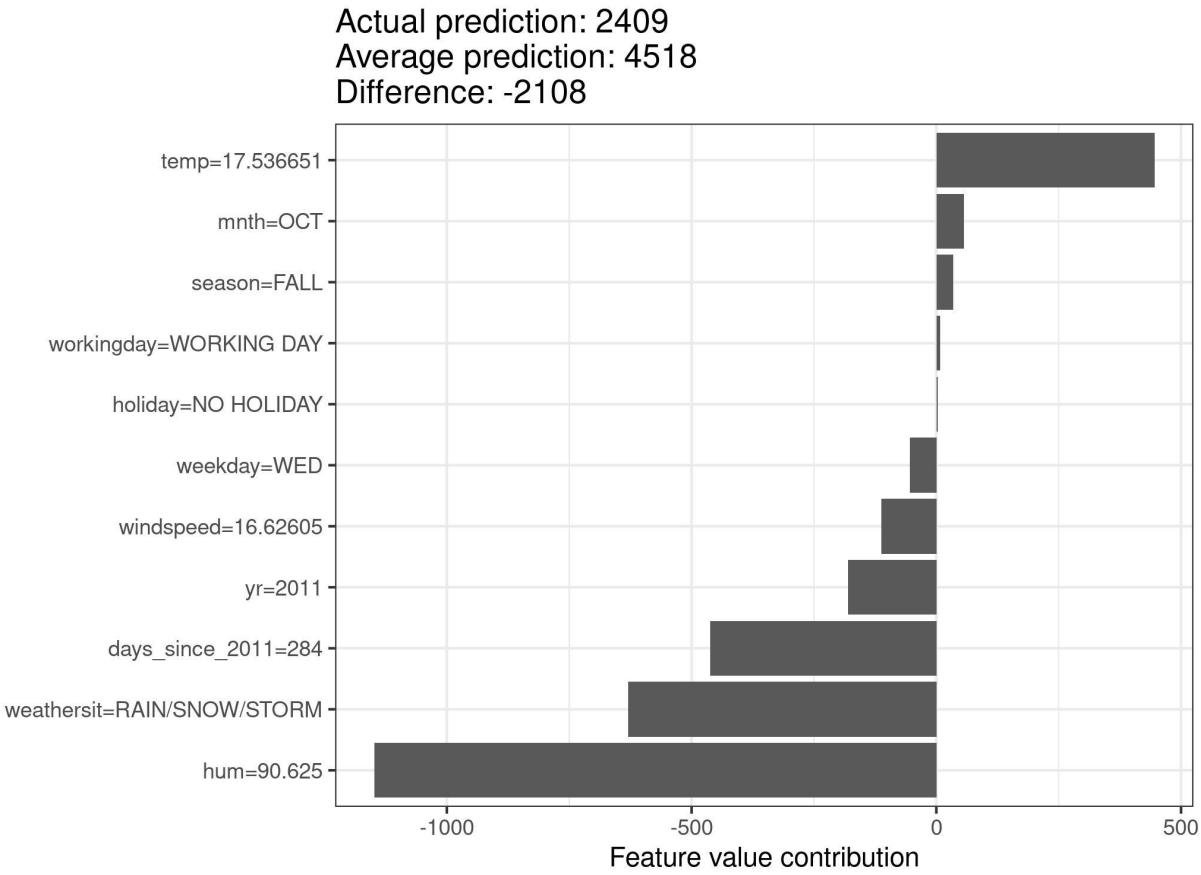
Examples and Interpretation

- The interpretation of the Shapley value for feature value j is: The value of the j-th feature contributed ϕj to the prediction of this particular instance compared to the average prediction for the dataset.
- The Shapley value works for both classification (if we are dealing with probabilities) and regression.
- **Classification:** We use the Shapley value to analyze the predictions of a random forest model predicting cervical cancer:



Shapley values for a woman in the cervical cancer dataset. With a prediction of 0.57, this woman's cancer probability is 0.54 above the average prediction of 0.03. The number of diagnosed STDs increased the probability the most. The sum of contributions yields the difference between actual and average prediction (0.54).

- **Regression:** For the bike rental dataset, we also train a random forest to predict the number of rented bikes for a day, given weather and calendar information. The explanations created for the random forest prediction of a particular day:



Shapley values for day 285. With a predicted 2409 rental bikes, this day is -2108 below the average prediction of 4518. The weather situation and humidity had the largest negative contributions. The temperature on this day had a positive contribution. The sum of Shapley values yields the difference of actual and average prediction (-2108).

- Be careful to interpret the Shapley value correctly: The Shapley value is the average contribution of a feature value to the prediction in different coalitions. The Shapley value is NOT the difference in prediction when we would remove the feature from the model.

Disadvantages

- The Shapley value **can be misinterpreted**. The Shapley value of a feature value is not the difference of the predicted value after removing the feature from the model training.
- The Shapley value is the wrong explanation method if you seek sparse explanations (explanations that contain few features). Explanations created with the Shapley value method **always use all the features**. Humans prefer selective explanations, such as those produced by LIME. LIME might be the better choice for explanations lay-persons have to deal with. Another solution is SHAP introduced by Lundberg and Lee (2016)65,

which is based on the Shapley value, but can also provide explanations with few features.
- **This means it cannot be used to make statements about changes in prediction for changes in the input**, such as: “If I were to earn €300 more a year, my credit score would increase by 5 points.”
- Another disadvantage is that **you need access to the data** if you want to calculate the Shapley value for a new data instance. It is not sufficient to access the prediction function because you need the data to replace parts of the instance of interest with values from randomly drawn instances of the data. This can only be avoided if you can create data instances that look like real data instances but are not actual instances from the training data.
- Like many other permutation-based interpretation methods, the Shapley value method suffers from **inclusion of unrealistic data instances when features are correlated.**

Code and tools

- Shapley values are implemented in both the `iml` and fastshap packages for R. In Julia, you can use Shapley.jl.

6. SHAP (SHapley Additive exPlanations)

- 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.

Example:

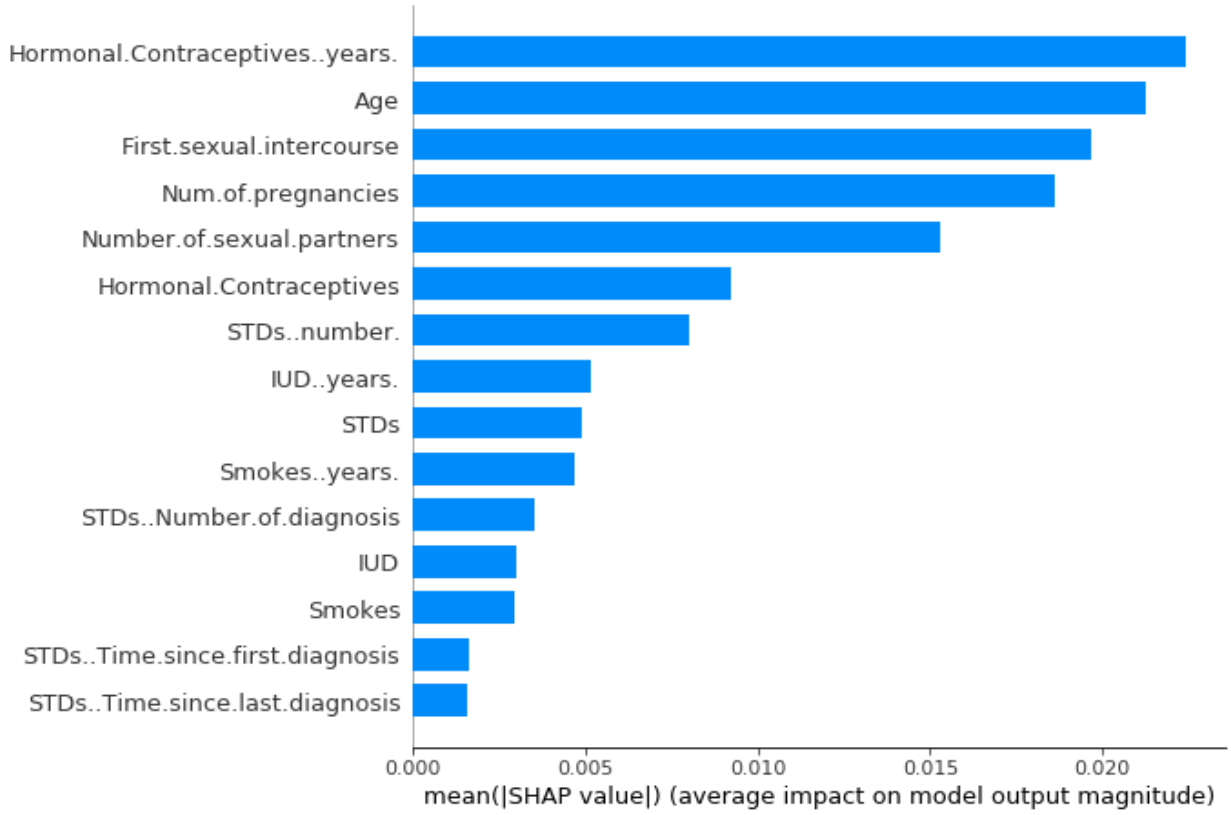
- Results shown after Trained a random forest classifier with 100 trees to predict the risk for cervical cancer. We will use SHAP to explain individual predictions. We can use the fast TreeSHAP estimation method instead of the slower KernelSHAP method, since a random forest is an ensemble of trees. But instead of relying on the conditional distribution, this example uses the marginal distribution.
- 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:

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.

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
- 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.

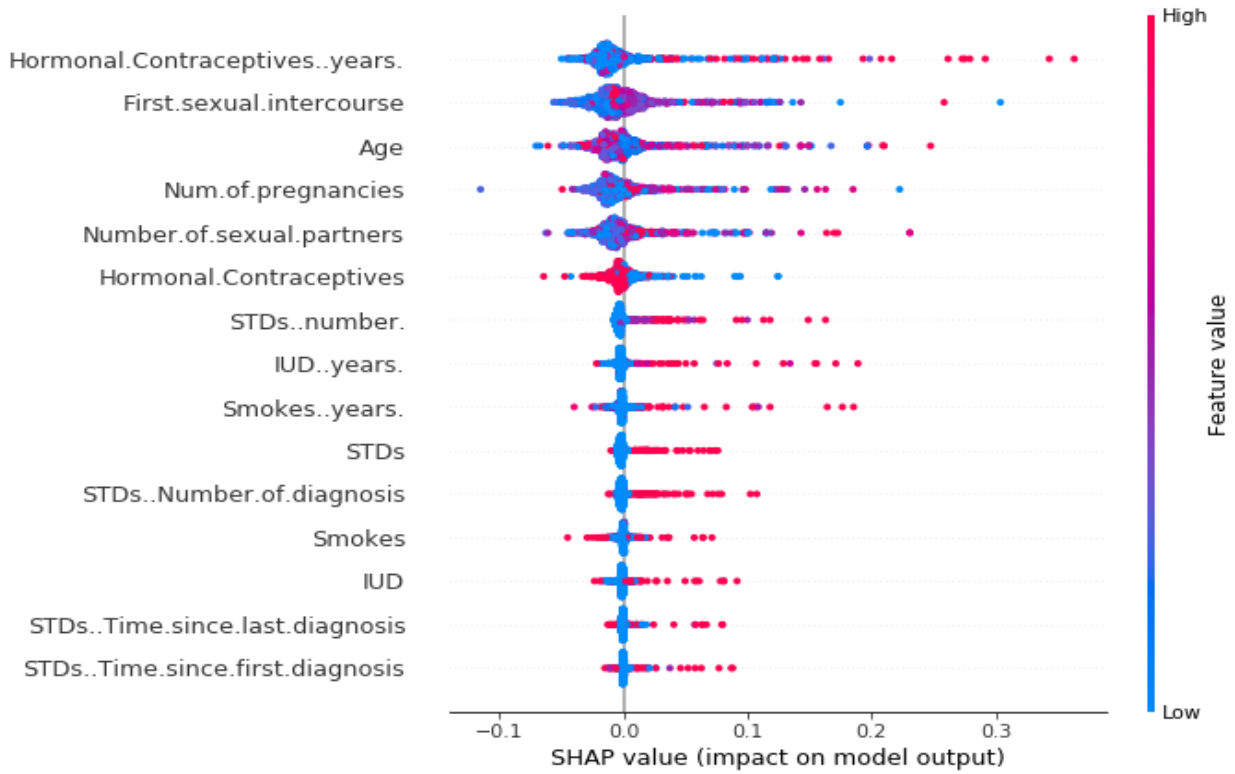


- SHAP feature importance measured as the mean absolute Shapley values. The number of years with hormonal contraceptives was the most important feature, changing the predicted absolute cancer probability on average by 2.4 percentage points (0.024 on

- Permutation feature importance is based on the decrease in model performance. SHAP is based on magnitude of feature attributions.

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.



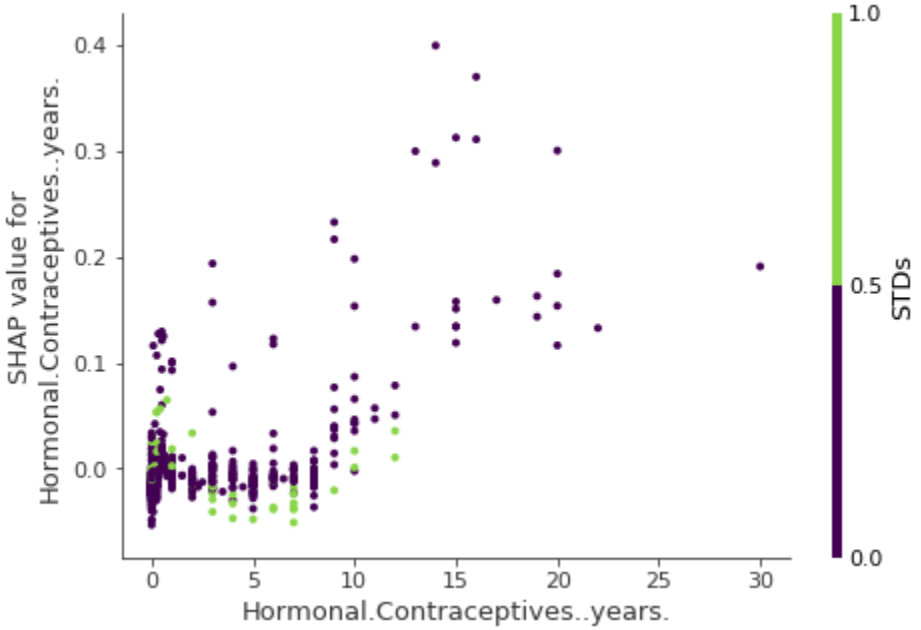
SHAP summary plot. Low number of years on hormonal contraceptives reduce the predicted cancer risk, a large number of years increases the risk. Your regular reminder:

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.

SHAP Interaction Values

- The interaction effect is the additional combined feature effect after accounting for the individual feature effects. 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:



SHAP features dependence plot with interaction visualization. Years on hormonal contraceptives interact with STDs. In cases close to 0 years, the occurrence of a STD increases the predicted cancer risk. For more years on contraceptives, the occurrence 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).

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.
- 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.

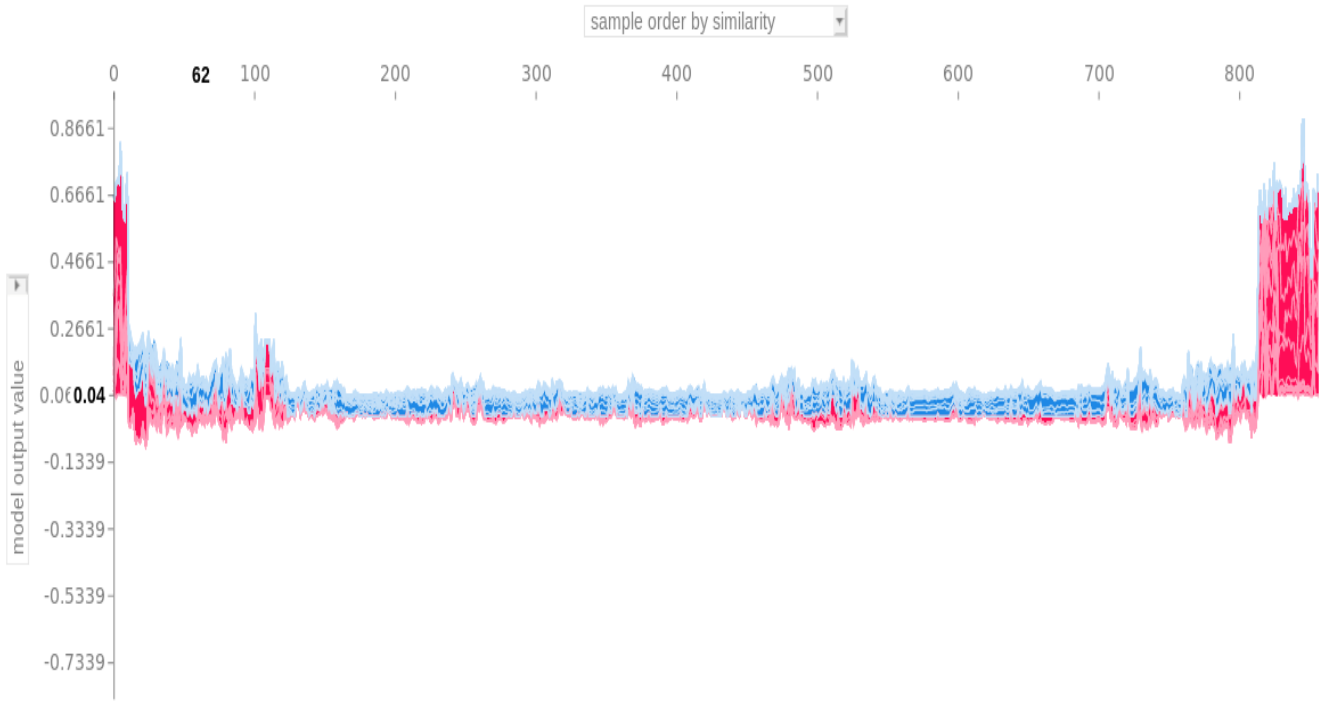


FIGURE 9.29: Stacked SHAP explanations clustered by explanation similarity. Each position on the x-axis is an instance of the data. Red SHAP values increase the prediction, blue values decrease it. One cluster stands out: On the right is a group with a high predicted cancer risk.

Disadvantage:

- 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. correlate**d, 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.
- 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 73. 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). **For the receivers of a SHAP explanation, it is a disadvantage: they cannot be sure about the truthfulness of the explanation.**

Code:

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 are the shapper and fastshap packages. SHAP is also included in the R xgboost package.

- https://github.com/slundberg/shap
- https://towardsdatascience.com/shap-explain-any-machine-learning-model-in-python-24207127cad7
- https://analyttica.com/understanding-shap-xai-through-leaps/#:~:text=SHAP%20stands%20for%20SHapley%20Additive,value%20over%20all%20possible%20coalitions.