A Brief Overview of PDP, LIME, and SHAP and Possible Ways to Fool Them for Tabular Data based on The Literature

Alireza Bayat Makou, Jamal Ahmad Bhatti, Danik Hollatz

Leibniz University of Hannover - Winter 2022

* All the uncited parts of this document are from the book:

Molnar, Christoph. "Interpretable machine learning.
A Guide for Making Black Box Models Explainable", 2019.
https://christophm.github.io/interpretable-ml-book/

Global Model-Agnostic Methods

- Global methods describe the average behavior of a machine learning model.
- The counterpart to global methods are local methods.
- Global methods are often expressed as expected values based on the distribution of the data.
- For example, the partial dependence plot, a feature effect plot, is the expected prediction when all other features are marginalized out.
- Since global interpretation methods describe average behavior, they are particularly useful when the modeler wants to understand the general mechanisms in the data or debug a model.

Partial Dependence Plot (PDP)

(J. H. Friedman 2001) - Friedman, Jerome H. “Greedy function approximation: A gradient boosting machine.” Annals of statistics (2001): 1189-1232.

The partial dependence plot (short PDP or PD plot) shows the marginal effect that one or two features have on the predicted outcome of a machine learning model.

The partial dependence plot is a global method: The method considers all instances and gives a statement about the global relationship of a feature with the predicted outcome.

A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic, or more complex.

- For example, when applied to a linear regression model, partial dependence plots always show a linear relationship.

$$\hat{f}_S(x_S) = E_{X_C}\left[\hat{f}(x_S, X_C)\right] = \int \hat{f}(x_S, X_C) d\mathbb{P}(X_C)$$

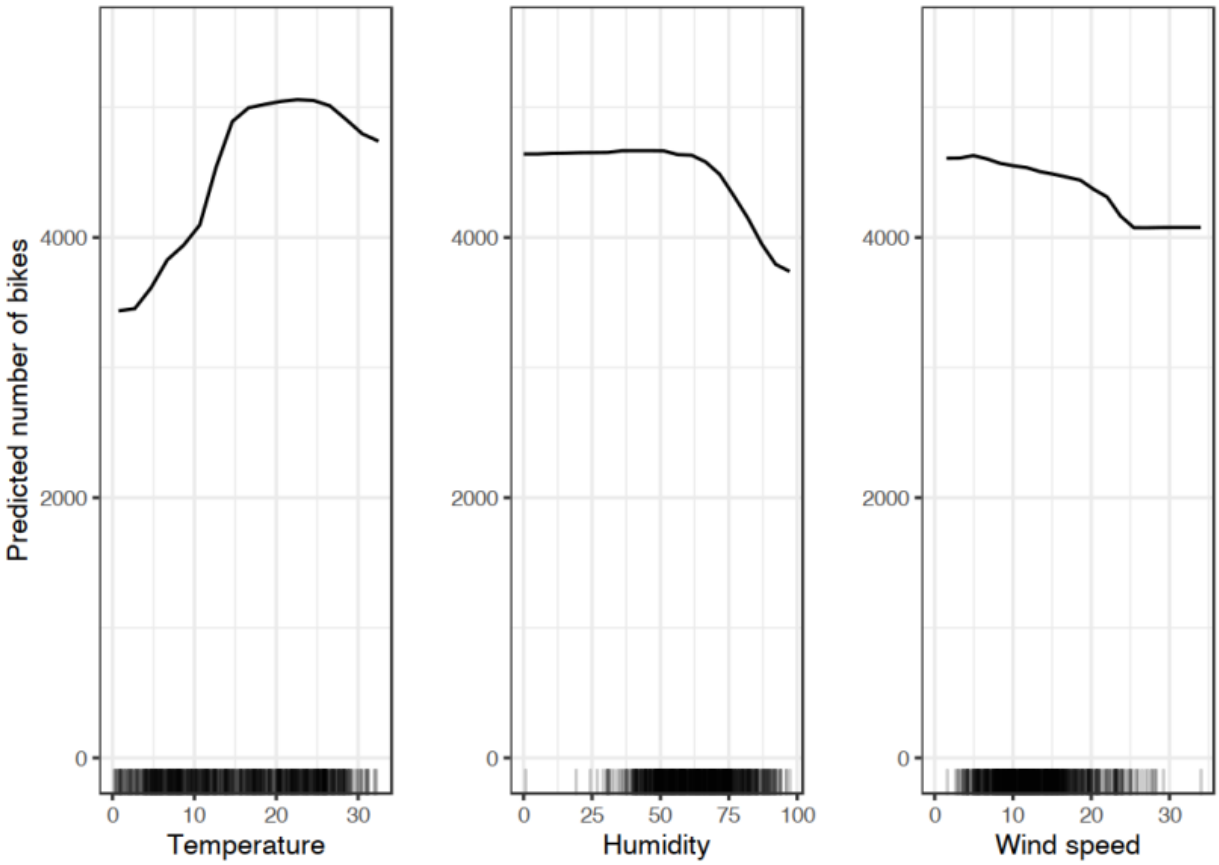
The x_S are the features for which the partial dependence function should be plotted and XC are the other features used in the machine learning model f, which are here treated as random variables.

The partial function (f hat) is estimated by calculating averages in the training data, also known as the Monte Carlo method:

$$\hat{f}_S(x_S) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)})$$

A flat PDP indicates that the feature is not important, and the more the PDP varies, the more important the feature is. (Greenwell et. al (2018))

For numerical features, importance is defined as the deviation of each unique feature value from the average curve. (Greenwell et. al (2018))



Advantages

The computation of partial dependence plots is **intuitive**.

In the uncorrelated case, the **interpretation is clear.**

- The partial dependence plot shows how the average prediction in your dataset changes when the j-th feature is changed.

Partial dependence plots are **easy to implement**.

The calculation for the partial dependence plots has a **causal interpretation**. (Zhao et al. 2017)

- We intervene on a feature and measure the changes in the predictions. In doing so, we analyze the causal relationship between the feature and the prediction.

- The relationship is causal for the model because we explicitly model the outcome as a function of the features but not necessarily for the real world!

Disadvantages

The **assumption of independence** is the biggest issue with PD plots.

- It is assumed that the feature(s) for which the partial dependence is computed are not correlated with other features.

- When the features are correlated, we create new data points in areas of the feature distribution where the actual probability is very low (for example it is unlikely that someone is 2 meters tall but weighs less than 50 kg).

- One solution to this problem is Accumulated Local Effect plots or short ALE plots that work with the conditional instead of the marginal distribution.

The realistic **maximum number of features** in a partial dependence function is two.

- This is not the fault of PDPs, but of the 2-dimensional representation (paper or screen) and also of our inability to imagine more than 3 dimensions.

Some PD plots do not show the **feature distribution**.

- Omitting the distribution can be misleading because you might overinterpret regions with almost no data.

- This problem is easily solved by showing a rug (indicators for data points on the x-axis) or a histogram.

Heterogeneous effects might be hidden because PD plots only show the average marginal effects.

- Suppose that for a feature half your data points have a positive association with the prediction – the larger the feature value the larger the prediction – and the other half has a negative association – the smaller the feature value the larger the prediction.

- The PD curve could be a horizontal line since the effects of both halves of the dataset could cancel each other out. You then conclude that the feature has no effect on the prediction.

- By plotting the individual conditional expectation curves instead of the aggregated line, we can uncover heterogeneous effects.

Fooling PDP (Baniecki, H., Kretowicz, W., & Biecek, P. (2021))

Baniecki, H., Kretowicz, W., & Biecek, P. (2021). Fooling Partial Dependence via Data Poisoning. *ArXiv, abs/2105.12837*.

This paper presents techniques for attacking Partial Dependence (plots, profiles, PDP), which are among the most popular methods of explaining any predictive model trained on tabular data.

The fooling is performed via poisoning the data to bend and shift explanations in the desired direction using genetic and gradient algorithms.

The novel approach of using a genetic algorithm for doing so is highly transferable as it generalizes both ways: in a model-agnostic and an explanation-agnostic manner

When considering an explanation as a function of model and data, there is a possibility to change one of these variables to achieve a different result (Zhao and Hastie, 2019).

Since this paper focuses on global-level explanations; instead, the results will modify a view of overall model behaviour, not specific to a single data point or image.

Authors aim to change the underlying dataset used to produce the model's explanation in a way to achieve the desired change in the PD.

In practice, machine learning practitioners compute such explanations using their estimators where a significant simplification may occur.

Thus, a slight shift of the dataset used to calculate PD may lead to unexpected or unintended results.

Authors approach attacking PD as an optimization problem for given criteria of attack efficiency, later called the attack loss that aims to change the output of a global-level explanation via data poisoning instead.

We exploit the explanation weaknesses concerning data distribution and causal inference by utilizing two ways of optimizing the loss:

Genetic-based

- algorithm that does not make any assumption about the model's structure – the black-box path from data inputs to the output prediction; thus, is model-agnostic.
- Further, we posit that for a vast number of explanations, clearly post-hoc global-level, the algorithm does not make assumption about their structure either; thus, becomes explanation-agnostic.

Gradient-based

- algorithm that is specifically designed for models with differentiable outputs, e.g. neural networks (Dombrowski et al., 2019; Dimanov et al., 2020).

We discuss and evaluate two possible strategies to perform the fooling:

Targeted attack

- changes the dataset to achieve the closest explanation result to the predefined desired function (Dombrowski et al., 2019; Heo et al., 2019).

Robustness check

- aims to achieve the most distant model explanation from the original one by changing the dataset, which corresponds to the sanity check (Adebayo et al., 2018)

Genetic-based Algorithm

Fooling PD in both strategies include a similar genetic algorithm.

The main idea is defining an individual as an instance of the dataset, iteratively perturb its values to achieve the desired explanation target, or perform the robustness check to observe the change.

These individuals are initialized with a value of the original dataset X' to form a population.

Subsequently, the initialization ends with mutating the individuals using a higher-than-default variance of perturbations.

Then, in each iteration, they are randomly crossed, mutated, evaluated with the attack loss, and selected based on the loss values.

Crossover swaps columns between individuals to produce new ones, which are then added to the population.

- The number of swapped columns can be randomized
- also the number of parents can be parameterized.

Mutation adds Gaussian noise to the individuals using scaled standard deviations of the variables.

- It is possible to constraint the change in data into the original range of variable values; also keep some variables unchanged.

Evaluation calculates the loss for each individual, which requires to compute model explanations for each dataset.

Selection reduces the number of individuals using rank selection, and elitism to guarantee several best individuals to remain into the next population.

Gradient-based Algorithms

This algorithm's main idea is to utilize a gradient descent to optimize the attack loss, considering the differentiability of the model's output with respect to input data.

Such assumption allows for a faster and more accurate convergence into a local minima using one of the stochastic optimizers; in our case, Adam (Kingma and Ba, 2015).

Note that the differentiability assumption is with respect to input data, not with respect to the model's parameters.

- Although we specifically consider the usage of neural networks because of their strong relation to differentiation, the algorithm's theoretical derivation does not require this type of model.

The gradient-based algorithm is similar to the genetic-based algorithm in that we aim to iteratively change the dataset used to calculate the explanation; nevertheless, its main assumption is that the model provides an interface for differentiation of output in respect to the input.

It is important to initialize the algorithm randomly, e.g. by adding Gaussian noise to data, especially in the robustness check.

- Otherwise, the difference between the original explanation and the changed one equals 0; thus, its derivative equals 0.

Datasets

Friedman dataset

- Regression problem (Friedman, 2001) where inputs X are independent variables uniformly distributed on the interval [0; 1], while the target y is created according to the formula: $y(X) = 10 \sin(\pi \cdot X1 \cdot X2) + 20(X3 - 0{,}5)^2 + 10X4 + 5X5$

- Only 5 variables are actually used to compute y, while the remaining are independent of.

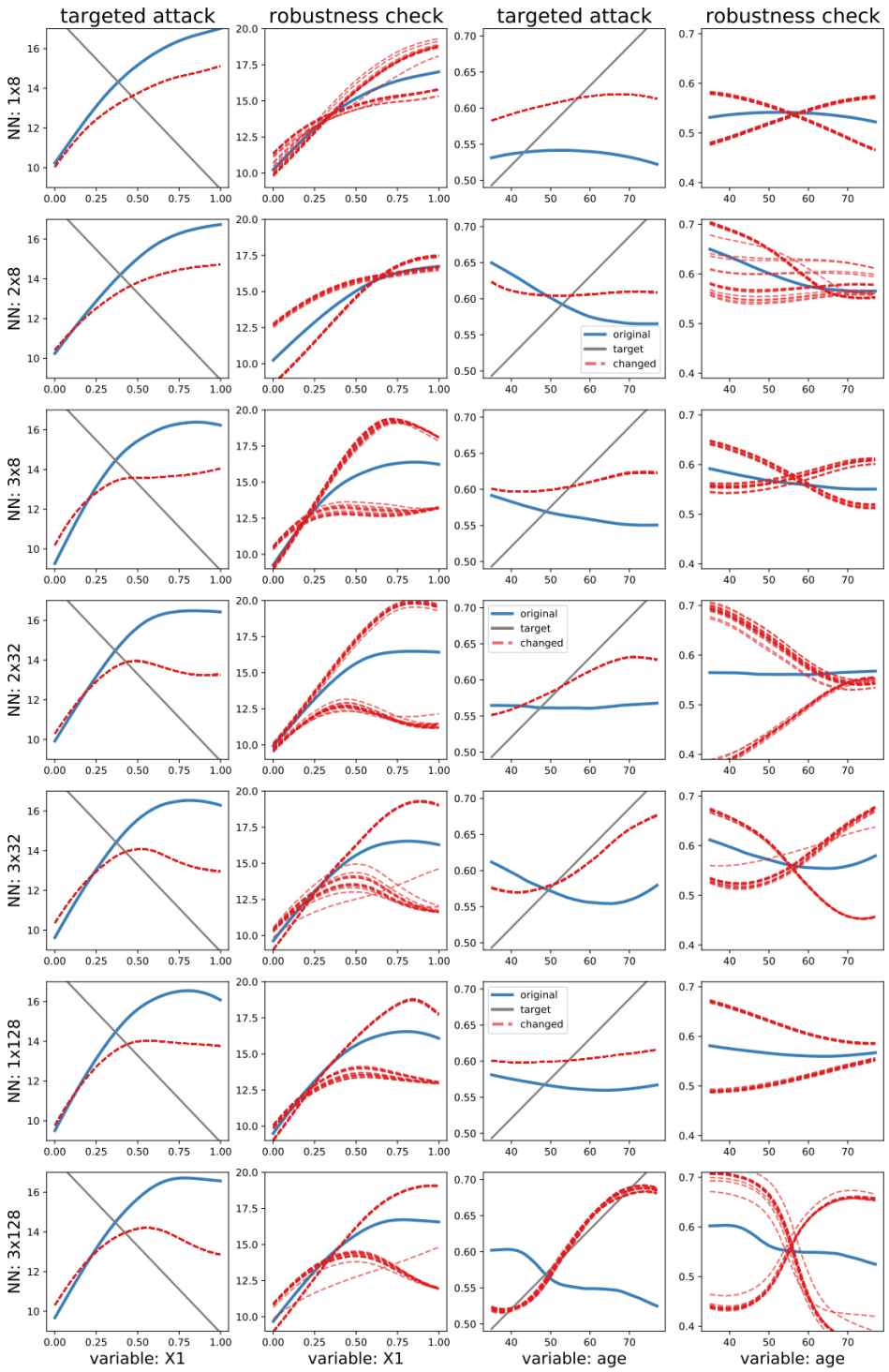
- We refer to this dataset as friedman and target explanations of the variable X1.

Heart dataset

- Classification task from UCI (Dua and Graff, 2017) has 303 observations, 5 continuous variables, 8 discrete variables, and an evenly-distributed binary target.

- We refer to this dataset as heart and target explanations of the variable age.

- Additionally, we set 8 categorical variables as constant during the performed attack algorithms because we mainly rely on incremental change in the values of continuous variables, and categorical variables are out of the scope of this work.



- Fooling Partial Dependence of neural network models (rows) fitted to the friedman and heart datasets.
- Authors performed multiple randomly initiated gradient-based attacks on the explanations of variables X1 and age respectively.
- The blue line denotes the original explanation, the red lines are the explanations after the attack, and in the targeted attack, the grey line denotes the desired target.
- Y axis denotes the model's architecture: layers×neurons.

Observations

Authors observe that PD explanations are especially vulnerable in complex models.

Authors found the explanations of NN, SVM and deep DT the most vulnerable to the fooling methods.

In contrast, RF seems to provide robust explanations.

Explanations of low-variance models prove to be robust to the attacks, while very complex models should not be explained with PD as the data poisoning easily manipulates these.

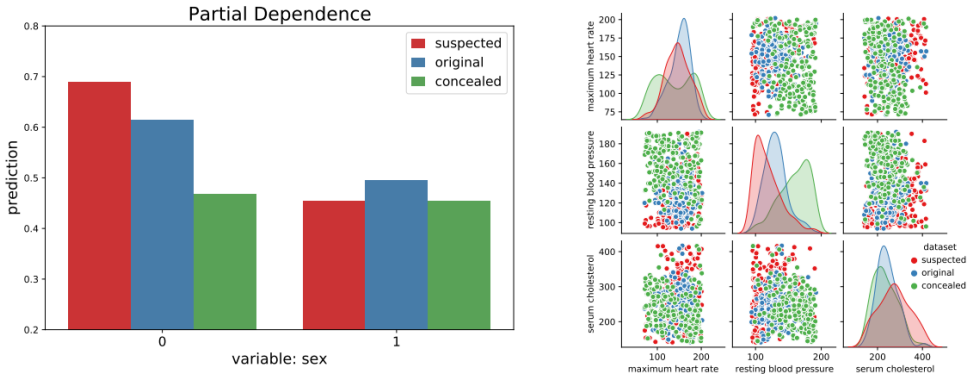


Figure 4: Partial Dependence of sex in the SVM model prediction of a heart attack (class 0). **Left:** Two manipulated explanations present a suspected or concealed variable contribution into the predicted outcome. **Right:** Distribution of the three poisoned variables from the data, in which sex and the remaining 9 variables attributing to the explanation are unchanged.

Scenarios

Adversarial scenario

Authors e consider three stakeholders apparent in explainable machine learning: developer, auditor, and prediction recipients.

Let us assume that the model predicting a heart attack should not take into account a patient's sex; although, it might be a valuable predictor.

An auditor analyses the model using Partial Dependence; therefore, the developer supplies a poisoned dataset for this task.

Figure 4 presents two possible outcomes of the model audit that are unequivocally bound to the explanation result and dataset.

The model remains unchanged while the stated assumption is concealed; thus, the prediction recipients become vulnerable.

Additionally, we supply an alternative scenario where the developer wants to provide evidence of model unfairness to raise suspicion.

Supportive scenario

Authors consider an equation of three variables: data, model, and explanation

Thus, they poison the data to fool the explanation while the model remains unchanged.

Authors present a moderate change in data distribution to introduce a concept of analysing such relationships for explanatory purposes, e.g. the first result might suggest that resting blood pressure and maximum heart rate contribute to the explanation of age;

the second result suggests how these variables contribute to the explanation of sex. We conclude that the data shift is worth exploring to analyse variable interactions in models.

Local Model-Agnostic Methods

- Local interpretation methods explain individual predictions.
- Local surrogate models (LIME) explain a prediction by replacing the complex model with a locally interpretable surrogate model.
- Shapley values is an attribution method that fairly assigns the prediction to individual features.
- SHAP is another computation method for Shapley values but also proposes global interpretation methods based on combinations of Shapley values across the data.
- LIME and Shapley values are attribution methods so that the prediction of a single instance is described as the sum of feature effects.

Local interpretable model-agnostic explanations (LIME)

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "Why should I trust you?: Explaining the predictions of any classifier." Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM (2016).

Local surrogate models are interpretable models that are used to explain individual predictions of black-box machine learning models.

Surrogate models are trained to approximate the predictions of the underlying black-box model.

Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.

LIME tests what happens to the predictions when you give variations of your data into the machine learning model.

1. LIME generates a new dataset consisting of perturbed samples and the corresponding predictions of the black-box model.
2. On this new dataset LIME then trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest.

The learned model should be a good approximation of the machine learning model predictions locally, but it does not have to be a good global approximation. (fidelity)

$$\text{explanation}(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

The user has to determine the complexity, e.g. by selecting the maximum number of features that the linear regression model may use.

The recipe for training local surrogate models:

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

Perturbation of Data

For text and images, the solution is to turn single words or super-pixels on or off.

In the case of tabular data, LIME creates new samples by perturbing each feature individually, drawing from a normal distribution with mean and standard deviation taken from the feature.

LIME for Tabular Data

LIME samples are not taken around the instance of interest, but from the training data's mass center, which is problematic.

But it increases the probability that the result for some of the sample points predictions differ from the data point of interest and that LIME can learn at least some explanation.

Defining a meaningful neighborhood around a point is difficult.

LIME currently uses an exponential smoothing kernel to define the neighborhood.

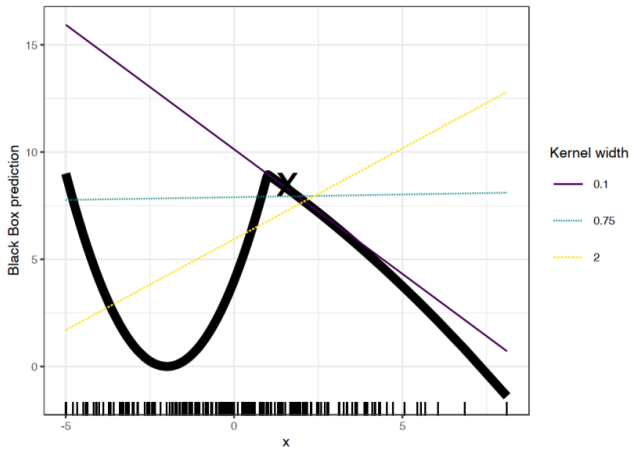
A smoothing kernel is a function that takes two data instances and returns a proximity measure.

The kernel width determines how large the neighborhood is.

A small kernel width means that an instance must be very close to influence the local model, a larger kernel width means that instances that are farther away also influence the model.

If you look at LIME's Python implementation (file lime/lime_tabular.py) you will see that it uses an exponential smoothing kernel (on the normalized data) and the kernel width is 0.75 times the square root of the number of columns of the training data.

The big problem is that we do not have a good way to find the best kernel or width.



It gets worse in high-dimensional feature spaces.

It is also very unclear whether the distance measure should treat all features equally.

Distance measures are quite arbitrary and distances in different dimensions (aka features) might not be comparable at all.

Advantages

Even if you **replace the underlying machine learning model**, you can still use the same local, interpretable model for the explanation.

Local surrogate models **benefit from the literature** and experience of training and interpreting interpretable models.

When using Lasso or short trees, the resulting explanations are short (= selective) and possibly contrastive. Therefore, they make **human-friendly explanations.**

LIME is one of the few methods that work for **tabular data, text, and images.**

The fidelity measure (how well the interpretable model approximates the black box predictions) gives us a good idea of how reliable the interpretable model is in explaining the black box predictions in the neighborhood of the data instance of interest.

LIME is implemented in Python and R and is **very easy to use.**

The explanations created with local surrogate models **can use other (interpretable) features** than the original model was trained on.

- A text classifier can rely on abstract word embeddings as features, but the explanation can be based on the presence or absence of words in a sentence.

- The regression model could be trained on components of a principal component analysis (PCA) of answers to a survey, but LIME might be trained on the original survey questions.

Disadvantages

The **correct definition of the neighborhood** is a very big, unsolved problem when using LIME with tabular data.

- For each application, you have to try different kernel settings and see for yourself if the explanations make sense.

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.

The **complexity** of the explanation model has to be defined in advance.

Instability of the explanations.

LIME explanations **can be manipulated** by the data scientist to hide biases. (Slack et al. 2020)

Shapley Values

Shapley, Lloyd S. “A value for n-person games.” Contributions to the Theory of Games 2.28 (1953): 307-317.

A prediction can be explained by assuming that each feature value of the instance is a “player” in a game where the prediction is the payout.

Shapley values – a method from coalitional game theory – tells us how to fairly distribute the “payout” among the features.

The Shapley value, coined by Shapley (1953), 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.

- 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).

The Shapley value is the average marginal contribution of a feature value across 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.
- 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.

The Shapley value works for both classification (if we are dealing with probabilities) and regression.

Interpretation

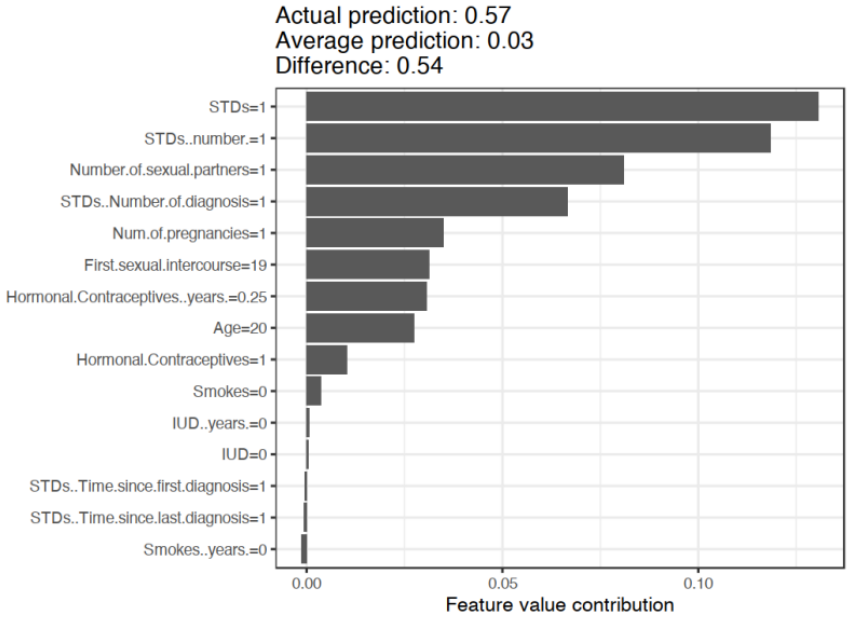
The interpretation of the Shapley value for feature value j is:

- The value of the j-th feature contributed ϕ_j to the prediction of this particular instance compared to the average prediction for the dataset.

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.



The sum of contributions yields the difference between actual and average prediction (0.54).

The Shapley Value

The Shapley value is a solution for computing feature contributions for single predictions for any machine learning model.

The Shapley value is defined via a value function val of players in S.

The Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations:

$$\phi_j(val) = \sum_{S \subseteq \{1,\dots,p\} \setminus \{j\}} \frac{|S|!\,(p-|S|-1)!}{p!} \left(val\left(S \cup \{j\}\right) - val(S)\right)$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained, and p is the number of features.

$val_x(S)$ is the prediction for feature values in set S that are marginalized over features that are not included in set S:

$$val_x(S) = \int \hat{f}(x_1, \dots, x_p) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X))$$

A concrete example:
The machine learning model works with 4 features x1, x2, x3, and x4 and we evaluate the prediction for the coalition S consisting of feature values x1 and x3:

$$val_x(S) = val_x(\{1,3\}) = \int_{\mathbb{R}} \int_{\mathbb{R}} \hat{f}(x_1, X_2, x_3, X_4) d\mathbb{P}_{X_2 X_4} - E_X(\hat{f}(X))$$

Shapley Value Properties

The Shapley value isthe only attributionmethodthat satisfiesthe properties Efficiency, Symmetry, Dummy and Additivity, which together can be considered a definition of a fair payout.

Efficiency

The feature contributions must add up to the difference ofprediction for x and the average.

$$\sum_{j=1}^{p} \phi_j = \hat{f}(x) - E_X(\hat{f}(X))$$

Symmetry

The contributions of two feature values j and k should be the same if they contribute equally to all possible coalitions. If

$$val(S \cup \{j\}) = val(S \cup \{k\})$$

For all: $S \subseteq \{1, \dots, p\} \quad \{j, k\}$

Then: $\phi_j = \phi_k$

Dummy

A feature j that does not change the predicted value – regardless of which coalition of feature values it is added to – should have a Shapley value of 0. If

$$val(S \cup \{j\}) = val(S)$$

For all: $S \subseteq \{1, \dots, p\}$

Then: $\phi_j = 0$

Additivity

For a game with combined payouts val+val+ the respective Shapley values are as follows:

$$\phi_j + \phi_j^+$$

Suppose you trained a random forest,which means that the prediction is an average of many decision trees.

- The Additivity property guarantees that for a feature value, you can calculate the Shapley value for each tree individually, average them, and get the Shapley value for the feature value for the random forest.

Estimating the Shapley Value

All possible coalitions (sets) of feature values have to be evaluated with and without the j-th feature to calculate the exact Shapley value.

For more than a few features, the exact solution to this problem becomes problematic as the number of possible coalitions exponentially increases as more features are added.

Strumbelj et al. (2014) propose an approximation with Monte-Carlo sampling:

$$\hat{\phi}_j = \frac{1}{M}\sum_{m=1}^{M}\left(\hat{f}(x^m_{+j}) - \hat{f}(x^m_{-j})\right)$$

1. where $\hat{f}(x^m_{+j})$ is the prediction for x, but with a random number of feature values replaced by feature values from a random data point z, except for the respective value of feature j.

2. The x-vector x^m_{-j} is almost identical to x^m_{+j}, but the value x^m_j is also taken from the sampled z.

3. Each of these M new instances is a kind of “Frankenstein Monster” assembled from two instances.

Approximate Shapley estimation for single feature value

- Output: Shapley value for the value of the j-th feature
- Required: Number of iterations M, instance of interest x, feature index j, data matrix X, and machine learning model f
  - For all m = 1,...,M:
    * Draw random instance z from the data matrix X
    * Choose a random permutation o of the feature values
    * Order instance x: $x_o = (x_{(1)}, \dots, x_{(j)}, \dots, x_{(p)})$
    * Order instance z: $z_o = (z_{(1)}, \dots, z_{(j)}, \dots, z_{(p)})$
    * Construct two new instances
      · With j: $x_{+j} = (x_{(1)}, \dots, x_{(j-1)}, x_{(j)}, z_{(j+1)}, \dots, z_{(p)})$
      · Without j: $x_{-j} = (x_{(1)}, \dots, x_{(j-1)}, z_{(j)}, z_{(j+1)}, \dots, z_{(p)})$
    * Compute marginal contribution: $\phi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j})$
- Compute Shapley value as the average: $\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m$

First, select an instance of interest x, a feature j and the number of iterations M.

For each iteration, a random instance z is selected from the data and a random order of the features is generated.

Two new instances are created by combining values from the instance of interest x and the sample z.

The instance x_{+j} is the instance of interest, but all values in the order after feature j are replaced by feature values from the sample z.

The instance x_{-j} is the same as x_{+j}, but in addition has feature j replaced by the value for feature j from the sample z.

The difference in the prediction from the black box is computed:

$$\phi_j^m = \hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m)$$

All these differences are averaged and result in:

$$\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m$$

Averaging implicitly weighs samples by the probability distribution of X.

The procedure has to be repeated for each of the features to get all Shapley values.

Advantages

The difference between the prediction and the average prediction is **fairly distributed** among the feature values of the instance – the Efficiency property of Shapley values.

- This property distinguishes the Shapley value from other methods such as LIME.
- LIME does not guarantee that the prediction is fairly distributed among the features.
- The Shapley value might be the only method to deliver a full explanation.
- In situations where the law requires explainability – like EU's "right to explanations" – the Shapley value might be the only legally compliant method, because it is based on a solid theory and distributes the effects fairly.

The Shapley value allows **contrastive explanations.**

- Instead of comparing a prediction to the average prediction of the entire dataset, you could compare it to a subset or even to a single data point.
- This contrastiveness is also something that local models like LIME do not have.

The Shapley value is the only explanation method with a **solid theory**.

- The axioms – efficiency, symmetry, dummy, additivity – give the explanation a reasonable foundation.
- Methods like LIME assume linear behavior of the machine learning model locally, but there is no theory as to why this should work.

It's fancy because of game theoretical approach :D

Disadvantages

The Shapley value requires **a lot of computing time**.

- In 99.9% of real world problems, only the approximate solution is feasible.
- An exact computation of the Shapley value is computationally expensive because there are 2^k possible coalitions of the feature values and the "absence" of a feature has to be simulated by drawing random instances, which increases the variance for the estimate of the Shapley values estimation.
- The exponential number of the coalitions is dealt with by sampling coalitions and limiting the number of iterations M.
    - Decreasing M reduces computation time, but increases the variance of the Shapley value.
    - There is no good rule of thumb for the number of iterations M.
    - M should be large enough to accurately estimate the Shapley values, but small enough to complete the computation in a reasonable time.
    - It should be possible to choose M based on Chernoff bounds, but I have not seen any paper on doing this for Shapley values for machine learning predictions.

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 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.

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 SHAP36 introduced by Lundberg and Lee (2016), which is based on the Shapley value, but can also provide explanations with few features.

The Shapley value returns a simple value per feature, but **no prediction model** like LIME.

- 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.

- To simulate that a feature value is missing from a coalition, we marginalize the feature.
- This is achieved by sampling values from the feature’s marginal distribution.
- This is fine as long as the features are independent.
- When features are dependent, then we might sample feature values that do not make sense for this instance.
- But we would use those to compute the feature’s Shapley value.
    - One solution might be to permute correlated features together and get one mutual Shapley value for them.
    - Another adaptation is conditional sampling:
        - Features are sampled conditional on the features that are already in the team.
        - While conditional sampling fixes the issue of unrealistic data points, a new issue is introduced:
        - The resulting values are no longer the Shapley values to our game, since they violate the symmetry axiom, as found out by Sundararajan et. al (2019) and further discussed by Janzing et. al (2020).

Alternatives

SHAP is an alternative estimation method for Shapley values.

Another approach is called breakDown, which is implemented in the breakDown R package.

BreakDown also shows the contributions of each feature to the prediction, but computes them step by step.

Let us reuse the game analogy:

- We start with an empty team, add the feature value that would contribute the most to the prediction and iterate until all feature values are added.
- How much each feature value contributes depends on the respective feature values that are already in the “team”, which is the big drawback of the breakDown method.
- It is faster than the Shapley value method, and for models without interactions, the results are the same.

SHAP (SHapley Additive exPlanations)

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

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.

SHAP comes with many global interpretation methods based on aggregations of Shapley values.

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.

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.

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.

In the coalition vector, an entry of 1 means that the corresponding feature value is “present” and 0 that it is “absent”.

The representation as a linear model of coalitions is a trick for the computation of the ϕ’s.

Shapley values are the only solution that satisfies properties of Efficiency, Symmetry, Dummy and Additivity.

- SHAP also satisfies these, since it computes Shapley values.

SHAP Properties

In the SHAP paper, you will find discrepancies between SHAP properties and Shapley properties. SHAP describes the following three desirable properties:

Local Accuracy (Shapley efficiency property)

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

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

Missingness (This property is not among the properties of the “normal” Shapley values)

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.

Lundberg calls it a “minor book-keeping property”.

A missing feature could – in theory – have an arbitrary Shapley value without hurting the local accuracy property, since it is multiplied with $x'_j = 0$.

The Missingness property enforces that missing features get a Shapley value of 0.

In practice this is only relevant for features that are constant.

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

Consistency

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.

Let $f_x(z') = f(h_x(z'))$ 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:

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

KernelSHAP

KernelSHAP estimates for an instance x the contributions of each feature value to the prediction.

KernelSHAP consists of 5 steps:

- Sample coalitions $z'_k \in \{0,1\}^M, \quad k \in \{1,\dots,K\}$ (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.

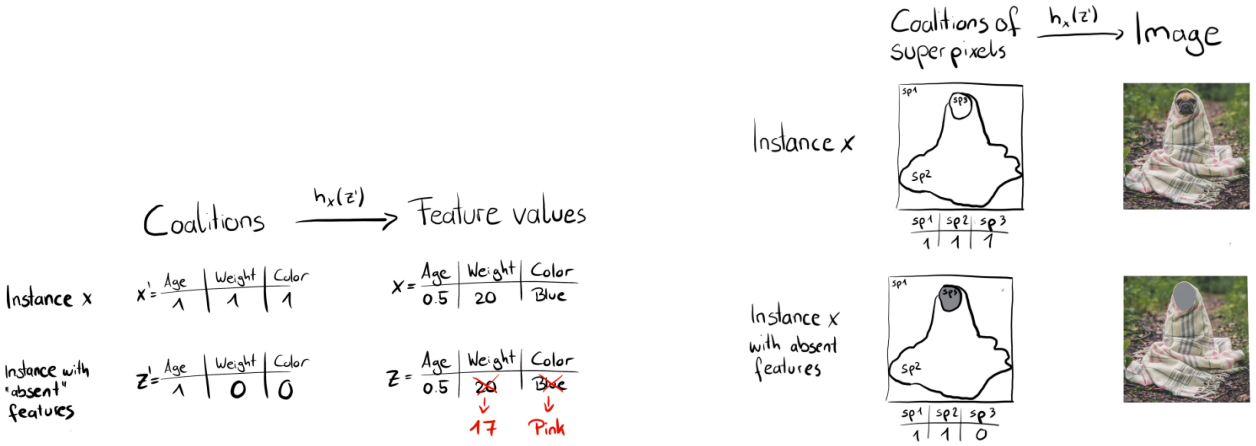
The function h_x maps 1's to the corresponding value from the instance x that we want to explain.

For tabular data, it maps 0's to the values of another instance that we sample from the data.

This means that we equate "feature value is absent" with "feature value is replaced by random feature value from data".

h_x for tabular data treats XC and XS 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 so 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.

The big difference to LIME is the weighting of the instances in the regression model.

LIME weights the instances according to how close they are to the original instance.

- The more 0's in the coalition vector, the smaller the weight in LIME.

SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation.

- Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights.

The intuition behind it is:

- We learn most about individual features if we can study their effects in isolation.
- If a coalition consists of a single feature, we can learn about the features' isolated main effect on the prediction.
- If a coalition consists of all but one feature, we can learn about this features' total effect (main effect plus feature interactions).
- If a coalition consists of half the features, we learn little about an individual features contribution, as there are many possible coalitions with half of the features.

To achieve Shapley compliant weighting, Lundberg et. al propose the SHAP kernel:

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

Here, M is the maximum coalition size and $|z'|$ the number of present features in instance z'.

Lundberg and Lee show that linear regression with this kernel weight yields Shapley values.

If you would use the SHAP kernel with LIME on the coalition data, LIME would also estimate Shapley values!

We can be a bit smarter about the sampling of coalitions:

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

We 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 isthetraining 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 ϕj'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)

TreeSHAP

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

Lundberg et. al (2018) 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 $E_{XS|XC}(f(x)|x_S)$ instead of the marginal expectation.

The problem with the conditional expectation is that feature that have no influence on the prediction function f can get a TreeSHAP estimate different from zero. (Sundararajan et al. 2019 / Janzing et al. 2019)

The non-zero estimate can happen when the feature is correlated with another feature that actually has an influence on the prediction.

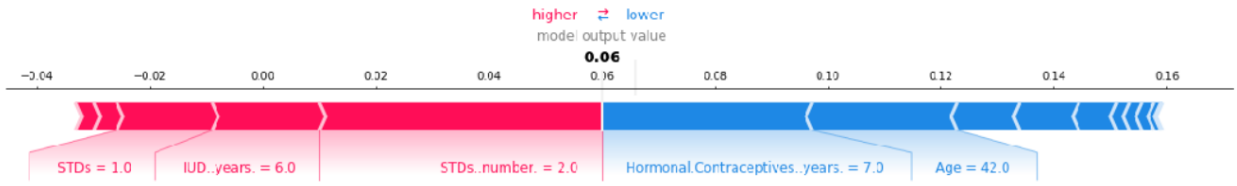
Compared to exact KernelSHAP, it reduces the computational complexity from $O(TL2^M)$ to $O(TLD^2)$, where T is the number of trees, L is the maximum number of leaves in any tree and D the maximal depth of any tree.

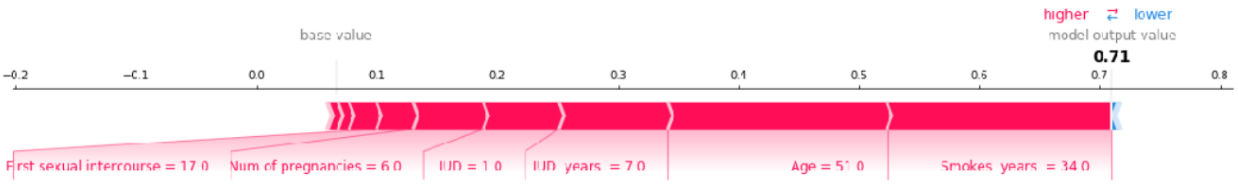
Different SHAP Plots

SHAP Force Plot

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

- 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.





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 and the Plot

Features with large absolute Shapley values are important.

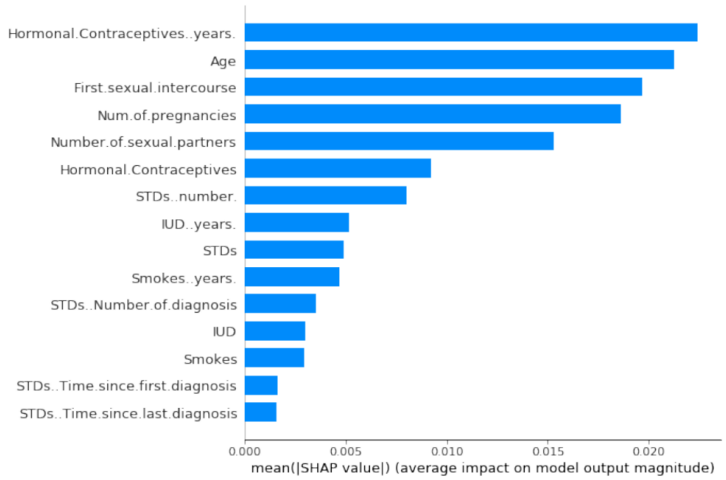
Since we want the global importance, we sum the absolute Shapley values per feature across the data:

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

Next, we sort the features by decreasing importance and plot them.

SHAP feature importance is an alternative to permutation feature importance. There is a big difference between both importance measures:

- Permutation Feature importance is based onthe decrease inmodel performance.
- SHAP is based on magnitude of feature attributions.



SHAP Summary Plot

The summary plot combines feature importance with feature effects.

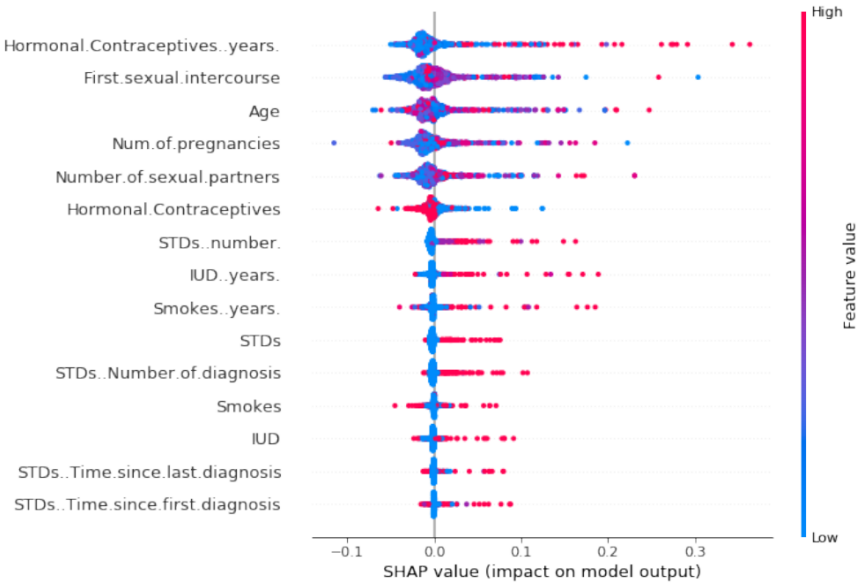
Each point on the summary plot is a Shapley value for a feature and an instance.

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 Dependence Plot

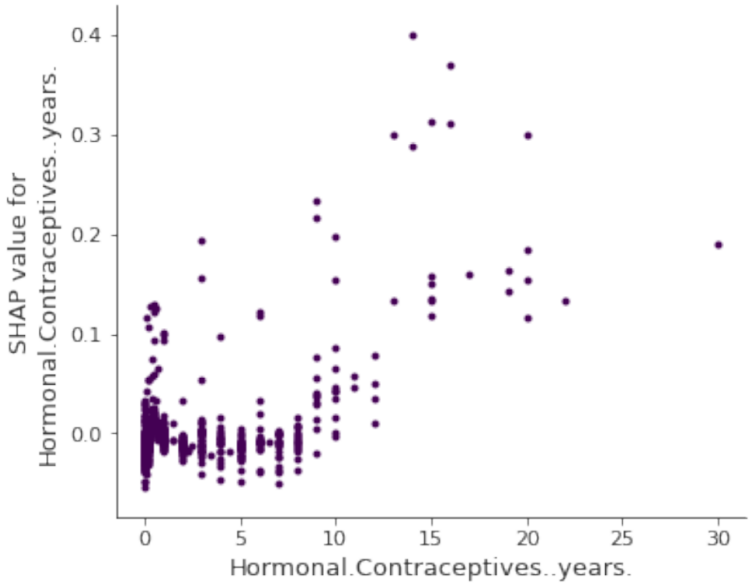
SHAP feature dependence might be the simplest global interpretationplot:

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.

SHAP dependence plots are an alternative to partial dependence plots and accumulated local effects.

- While PDP and ALE plot show average effects, SHAP dependence also shows the variance on the y-axis.
- Especially in case of interactions, the SHAP dependence plot will be much more dispersed in the y-axis.

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



SHAP Interaction Values

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

The Shapley interaction index from game theory is defined as:

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

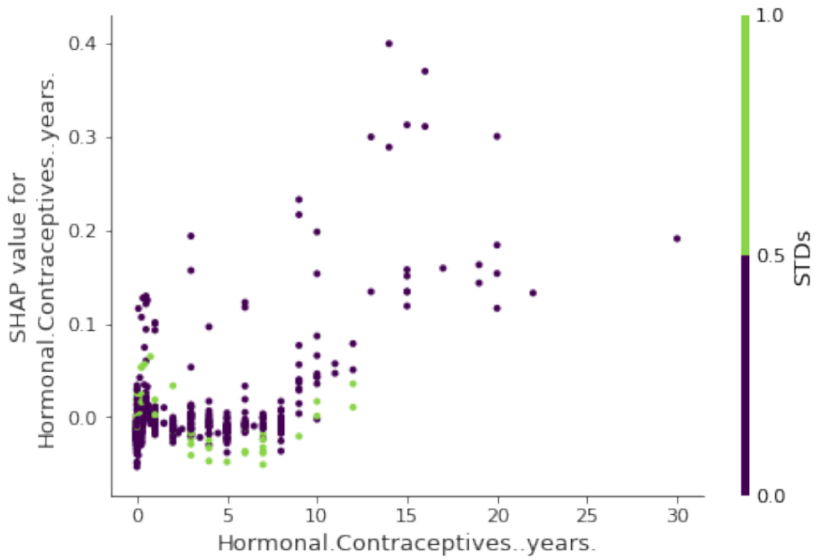
when $i \neq j$ and:

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

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 MxM, where M is the number of features.



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

Clustering SHAP values

You can cluster your data with the help of Shapley values.

The goal of clustering is to find groups of similar instances.

Normally, clustering is based on features.

Features are often on different scales.

For example, height might be measured in meters, color intensity from 0 to 100 and some sensor output between -1 and 1.

The difficulty is to compute distances between instances with such different, non-comparable features.

SHAP clustering works by clustering on Shapley values of each instance.

This means that you cluster instances by explanation similarity.

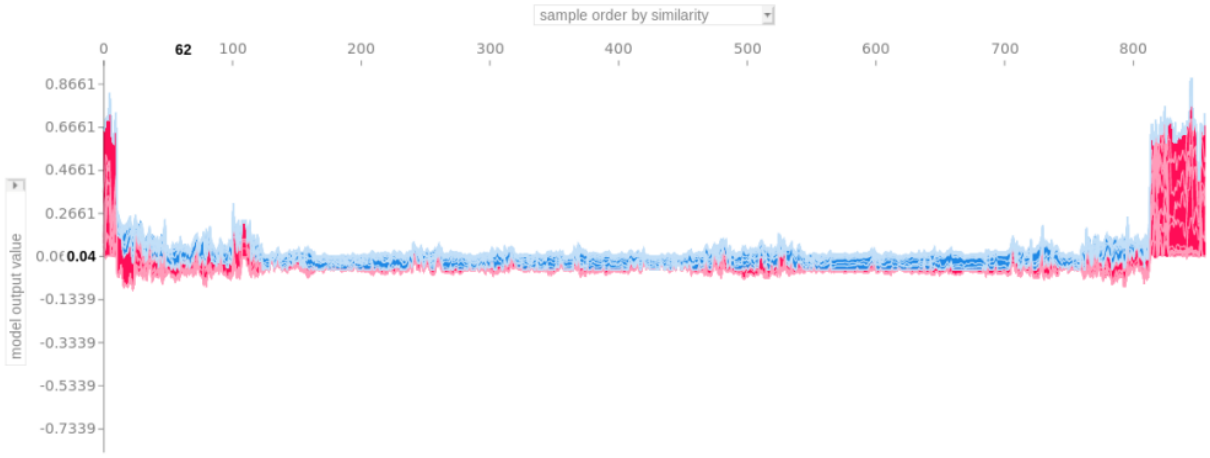
All SHAP values have the same unit – the unit of the prediction space.

You can use any clustering method.

The following example uses hierarchical agglomerative clustering to order the instances.

The plot consists of many force plots, each of which explains the prediction of an instance.

We rotate the force plots vertically and place them side by side according to their clustering similarity.



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

Advantages

Since SHAP computes Shapley values, **all the advantages of Shapley values** apply.

SHAP **connects LIME and Shapley values**.

SHAP has a f**ast implementation for tree-based models**.

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
- 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.

Disadvantages

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)

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.

It is possible to create **intentionally misleading interpretations** with SHAP, which can hide biases. (Slack et al. 2020)

Fooling LIME and SHAP (Slack, Dylan, et al. 2020)

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

In this paper, we demonstrate that post hoc explanations techniques that rely on input perturbations, such as LIME and SHAP, are not reliable.

Specifically, we propose a novel scaffolding technique that effectively hides the biases of any given classifier by allowing an adversarial entity to craft an arbitrary desired explanation.

Our approach can be used to scaffold any biased classifier in such a way that its predictions on the input data distribution still remain biased, but the post hoc explanations of the scaffolded classifier look innocuous.

These methods (LIME and SHAP) estimate the contribution of individual features towards a specific prediction by generating perturbations of a given instance in the data and observing the effect of these perturbations on the output of the black-box classifier.

Authors approach exploits the fact that post hoc explanation techniques such as LIME and SHAP are perturbation-based, to create a scaffolding around any given biased black box classifier in such a way that its predictions on input data distribution remain biased, but its behavior on the perturbed data points is controlled to make the post hoc explanations look completely innocuous.

Authors evaluate the effectiveness of the proposed framework on multiple real world datasets — COMPAS (Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. 2016), Communities and Crime (M Redmond. 2011. Communities and crime unnormalized data set. UCI Machine Learning Repository. (2011)), and German loan lending (Arthur Asuncion and David Newman. 2007. UCI machine learning repository, 2007.)

For each dataset, we craft classifiers that heavily discriminate based on protected attributes such as race (demographic parity ratio = 0), and show that our framework can effectively hide their biases.

Dataset	Size	Features	Positive Class	Sensitive Feature
COMPAS	6172	*criminal history, demographics, COMPAS risk score, jail and prison time*	High Risk (81.4%)	African-American (51.4%)
Communities & Crime	1994	*race, age, education, police demographics, marriage status, citizenship*	Violent Crime Rate (50%)	White Population (continuous)
German Credit	1000	*account information, credit history, loan purpose, employment, demographics*	Good Customer (70%)	Male (69%)

Input

The adversary provides the following to our framework:

- the biased classifier f which they intend to deploy in the real world and
- an input dataset X that is sampled from the real world input data distribution Xdist on which f will be applied.
    - Note that neither our framework nor the adversary has access to Xdist .

Output

The output of our framework will be a scaffolded classifier e (referred to as adversarial classifier henceforth) that behaves exactly like f when making predictions on instances sampled from Xdist , but will not reveal the underlying biases of f when probed with leading post hoc explanation techniques such as LIME and SHAP.

Since the feature importances output by LIME and SHAP rely heavily on perturbed instances (which may typically be OOD samples, e.g. Figure 1), the resulting explanations will make the classifier designed by the adversary look innocuous.

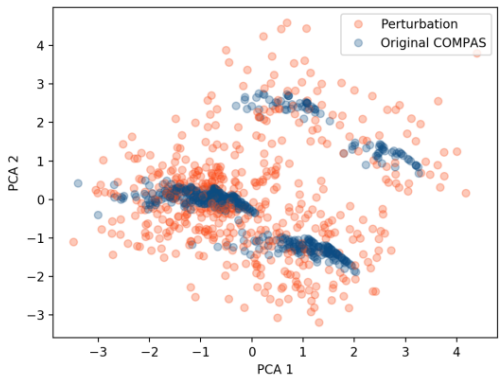


Figure 1: PCA applied to the COMPAS dataset (blue) as well as its LIME style perturbations (red). Even in this low-dimensional space, we can see that data points generated via perturbations are distributed very differently from instances in the COMPAS data. In this paper, we exploit this difference to craft adversarial classifiers.

Experimental Setup

Authors randomly split all the datasets into training (90%) and test (10%) sets.

Biased Classifier f

Authors construct f to be a perfectly discriminatory classifier that uses a sensitive feature i.e., f makes predictions purely based on race in the case of COMPAS and Communities and Crime datasets (if race = African American, then predict 1, otherwise set predict 0), and based on gender for the German Credit dataset.

Perturbations & OOD classifier

For LIME, authors generate perturbations for every data point in the data set X by adding random noise sampled from N(0, 1) to each feature value.

For SHAP, authors randomly choose a subset of features for each data point and mark their values as missing by replacing them with their corresponding values from background distribution.

Authors choose the background distribution for SHAP to be cluster centers from kmeans with 10 clusters.

Authors use these newly generated perturbations as well as the input data points to train the OOD classifier to detect OOD samples.

Authors use the standard sklearn implementation of a random forest with 100 tree estimators as our OOD classifier.

Unbiased Classifier ψ

Authors build the unbiased classifier ψ by constructing synthetic uncorrelated features that have zero correlation with sensitive attributes (e.g., race or gender).

Authors experiment with one or two uncorrelated features.

When there's only one uncorrelated feature in a particular experiment, ψ solely uses that to make predictions (if uncorrelated feature = 1, then predict 1, else predict 0).

On the other hand, when there are two uncorrelated features in an experiment, authors base the predictions on the xor of those two features.

Note that ψ does not have to be restricted to always use synthetic uncorrelated features. It can also use any other existing feature in the data to make predictions.

Authors experiment with synthetic uncorrelated features on COMPAS and CC dataset, and with Loan Rate % Income feature on the German credit dataset.

Generating Explanations

We use default LIME tabular implementation without discretization, and the default Kernel SHAP implementation with k-means with 10 clusters as the background distribution.

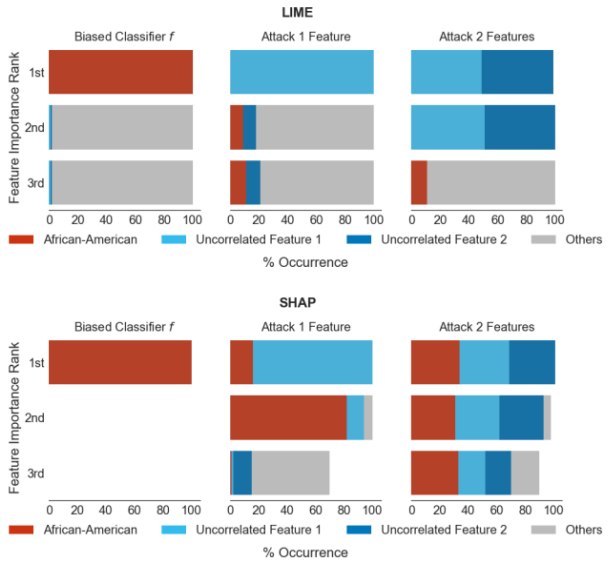


Figure 2: COMPAS: % of data points for which each feature (color coded) shows up in top 3 (according to LIME and SHAP's ranking of feature importances) for the biased classifier f (left), our adversarial classifier where ψ uses only one uncorrelated feature to make predictions (middle), and our adversarial classifier where ψ uses two uncorrelated features to make predictions (right).

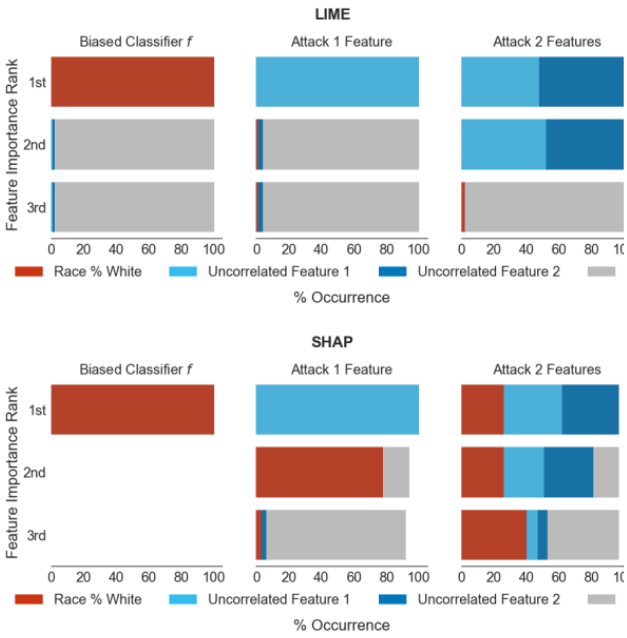


Figure 3: Communities and Crime: Similar to Fig 2; *Ra White* is the sensitive feature here.

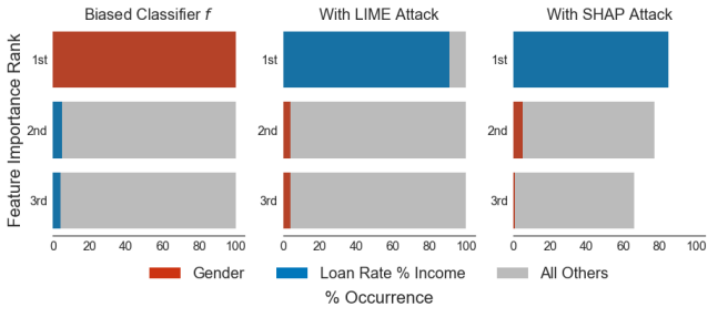


Figure 4: German credit: Similar to Fig 2 and 3, but unbiased classifier ψ uses an existing feature (*Loan Rate % Income*) to make predictions, and *Gender* is the sensitive feature. Feature importances for the biased classifier f shown in the figure (left) are generated using LIME; SHAP also produces similar feature importance values.

Evaluation

To evaluate how successful our attacks are on LIME and SHAP, authors compute the percentage of data points for which race, uncorrelated features (in case of COMPAS and CC) or Loan Rate % Income (in case of German credit data) show up in top 3 when features are ranked based on feature attributions output by LIME and SHAP.

Observations

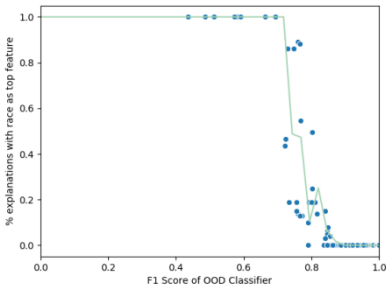
Results demonstrate that the LIME explanation technique has been effectively fooled by the adversarial classifier e for all three datasets.

SHAP's local accuracy property ensures that feature attributions must add up to the difference between a given prediction and the average prediction for the background distribution.

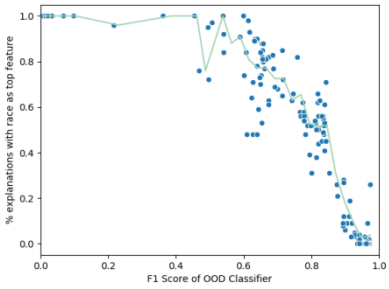
- This property will tend to distribute feature attributions across several features when it is not possible to identify a single most informative feature.

Attacks are effective as long as it is possible to differentiate between perturbed instances and input data points with a reasonable accuracy.

classifiers that are adversarial to LIME are ineffective against SHAP explanations, while adversarial classifiers designed for SHAP have varying effectiveness against LIME.



(a) LIME COMPAS Sensitivity Analysis



(b) SHAP COMPAS Sensitivity Analysis

Figure 5: Effectiveness of our attack: % of data points for which race is the most important feature vs. the F1 score of the OOD classifier (perturbation detection) for both LIME and SHAP on the COMPAS dataset over a variety of models (fit with line plot over mean of binned points).

Robustness to Hyperparameters

On varying the LIME kernel width between 0.15√M to 0.95√M (default kernel width is 0.75√M, where M is the number of features), the attacks remain equally effective.

Additionally, changing the LIME kernel to use l1 norm for distance instead of the default l2 norm also does not impact the effectiveness.

For SHAP, authors train the adversarial classifier assuming the default background distribution of K-means with 10 clusters; 5 and 15 clusters at test-time are also equally effective.

Using a background distribution of all zeroes for SHAP makes the attacks even more pronounced.

Related Works

Perturbation-based Explanation Methods

Perturbation-based methods are a popular tool for post hoc feature attribution.

In addition to LIME and SHAP, a number of other perturbation based techniques have been proposed in literature.

For instance, anchors (Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018) use (non-linear) rules to express more actionable local explanations.

GAM (Mark Ibrahim, Melissa Louie, Ceena Modarres, and John Paisley. 2019) interprets local attributions as conjoined weighted rankings and uses k-medoids clustering to identify prototypical explanations.

Criticism of Post hoc Explanations

Cynthia Rudin. (2019) argues that post hoc explanations are not reliable, as these explanations are not necessarily faithful to the underlying models and present correlations rather than information about the original computation.

Ghorbani et al. (Amirata Ghorbani, Abubakar Abid, and James Zou. 2019.) show that some explanation techniques can be highly sensitive to small perturbations in the input even though the underlying classifier's predictions remain unchanged.

Mittelstadt et al. (Brent Mittelstadt, Chris Russell, and Sandra Wachter. 2019) note that perturbation points created in LIME and SHAP are not at all intuitive, especially in case of structured data.

Adversarial Explanations

There has been some recent research on manipulating explanations in the context of image classification.

Dombrowski et al. (Ann-Kathrin Dombrowski, Maximilian Alber, Christopher J Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. 2019) show that by modifying inputs in a way that is imperceptible to humans, they can arbitrarily change saliency maps.

Heo et al. (Juyeon Heo, Sunghwan Joo, and Taesup Moon. 2019) also propose similar attacks on saliency maps.

Interpretability and Bias Detection

Doshi-Velez and Kim (Finale Doshi-Velez and Been Kim. 2017) argue that interpretability can help us evaluate if a model is biased or discriminatory.

On the other hand, Lipton (Zachary C. Lipton. 2018) posits that post hoc explanations can never definitively prove or disprove unfairness of any given classifier.

Selbst and Barocas (Andrew D Selbst and Solon Barocas. 2018) and Kroll et al. (Joshua A Kroll, Solon Barocas, Edward W Felten, Joel R Reidenberg, David G
Robinson, and Harlan Yu. 2016) show that even if a model is completely transparent, it is hard to detect and prevent bias due to the existence of correlated variables.

More recently, Aivodji et al. (Ulrich Aivodji, Hiromi Arai, Olivier Fortineau, Sébastien Gambs, Satoshi Hara, and Alain Tapp. 2019) demonstrated that post hoc explanations can potentially be exploited to fairwash i.e., rationalize decisions made by unfair black-box models.