Machine Learning in Bioinformatics

Explainability - XAI

Philipp Benner
philipp.benner@bam.de

VP.1 - eScience
Federal Institute of Materials Research and Testing (BAM)

February 8, 2026

Introduction

- Machine learning studies the relationship between
  - independent or predictor variables X
  - dependent or response variables Y
- Machine learning and statistics may have multiple goals [Zhao and Hastie, 2021]:
  - *Prediction:* Predict the response variables Y as accurate at possible from X
  - *Science:* If $X \rightarrow Y$ is a causal relationship, we may want to understand the *laws of nature* that determine this relationship

Introduction

- Two opposing cultures of statistical analysis [Breiman, 2001]:
  - *Data modeling culture:* Assume a parametric function f such that $Y = f(X) + \epsilon$, where ϵ models the aleatoric uncertainty. The parameters of f are often easy to interpret and the model is used to understand the laws of nature
  - *Algorithmic modeling culture:* Use of *black-box* models that are very complex and optimized to maximize predictive accuracy. Black-box models are notoriously difficult to interpret and do barely allow to draw any conclusions about the laws of nature
- If we have a black-box model, how can we still gain some interpretation?

Outline I

- Assume we have a black-box machine learning model f
- Can we gain some *limited* understanding of the predictions of f?
- Understanding the predictions increases our *trust* in f

Outline II

- Given a fixed input x, what is the *contribution* of each feature to the prediction $y = f(x)$?
  (Attribution Map / Saliency Maps)
  - Occlusion
  - Layer-wise relevance propagation (LRP) / DeepLIFT
  - Integrated gradients
  - Shapley values
  - SHAP
- Given a fixed input x, is there an interpretable model that approximates f locally?

Outline III

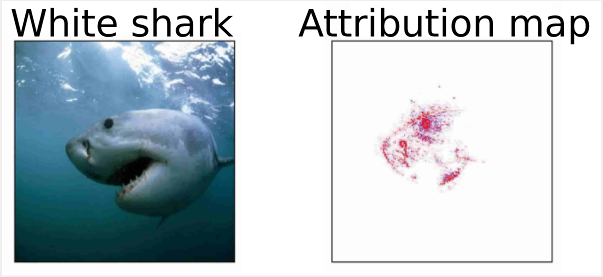
- Local interpretable model-agnostic explanations (LIME)
- Taylor approximations

- What would f predict if we vary one or more features?
  - Partial dependence plots (PDP)
  - Individual conditional expectation (ICE)
- What is the most likely input x for a given prediction $y = f(x)$?
  - Input optimization

Attribution maps

Attribution maps

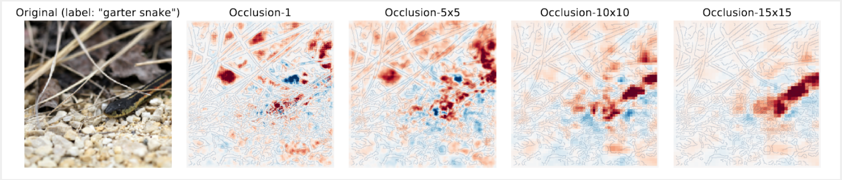
- Attribution maps are very popular with images, where the attribution of each pixel can be easily visualized
- Each input feature is assigned an attribution score (feature attribution)



[Kim et al., 2019]

Occlusion I

- Occlusion is a perturbation method that masks part of the input and measures the effect on the output of the network [Ancona et al., 2017]
- This method requires to evaluate the model for many perturbations
- The size of the mask is of particular importance

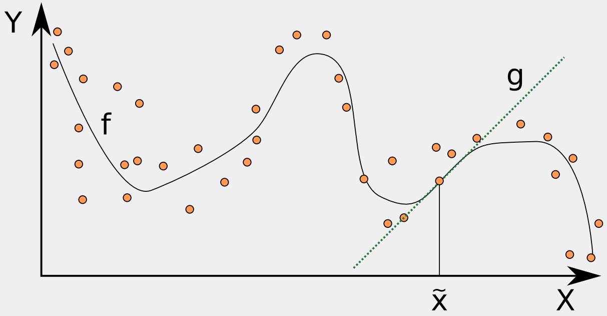


Occlusion II

- Assume we have a network that detects whether an image contains a cat
- If there are multiple cats in the image, occluding parts of the image with image patches (i.e. occluding at most one cat at a time) will not change the output of the network
- In this case, we would require masking multiple regions at the same time
- This leads to a combinatorial explosion

Gradient based explanations

Local explanations



Feature attribution

- Given a fixed input x and the corresponding output $y = f(x)$, what input features contribute most to the output value y?
- Note that for many applications (e.g. images) it is not very valuable to know which features (e.g. pixels) contribute most to the output of a neural network unless a specific input is considered
- The provided level of interpretability is hence limited to individual input data points

Gradient I

- Let f be a neural network or any other differentiable machine learning model

- Using the first-order Taylor expansion of f at an input $\tilde{x}$ we approximate f as a linear function

$$f(x) \approx f(\tilde{x}) + \nabla_x^\top f(\tilde{x})(x - \tilde{x})$$

  [Simonyan et al., 2013]

- With $w = \nabla_x f(\tilde{x})$ and $x' = x - \tilde{x}$ we obtain

$$f(x') \approx f(\tilde{x}) + w^\top x'$$

  where the gradient w can be easily interpreted as *feature importances*

Gradient × input I

- Using the gradient alone is problematic
- Let the network be defined as

$$f(x) = \max\{0, x - 10\}$$

  i.e. a single linear unit with ReLU activation

- The gradient is given by

$$\nabla_x f(x) = \begin{cases} 1 & \text{if } x > 10 \\ 0 & \text{otherwise} \end{cases}$$

- In this simple example, the larger x the larger the output $y = f(x)$ (assuming $x > 10$)

Gradient $\times$ input II

- However, for $f(20)$ we obtain the same attribution value as for $f(1000)$, i.e. 1 in both cases
- Multiplying the gradient with the input x seems to improve results [Shrikumar et al., 2016]
- For $f(20)$ we would obtain 20 as attribution value, whereas for $f(1000)$ the attribution is 1000

Integrated gradients I

- Integrated gradients (IG): Consider the gradient along an entire path from a baseline x_0 to an input $\tilde{x}$ [Sundararajan et al., 2017]

$$\mathrm{IG}_j(\tilde{x}) = (\tilde{x}^{(j)} - x_0^{(j)}) \int_{[0,1]} \frac{\partial f(\alpha \tilde{x} + (1-\alpha) x_0)}{\partial \tilde{x}^{(j)}} \mathrm{d}\alpha$$

- IG satisfies several convincing axioms that other methods violate

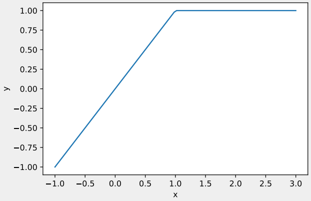
Integrated gradients II

Axiom 1: Sensitivity

Assume that x_0 and $\tilde{x}$ differ in at least one feature and that $f(x_0) \neq f(\tilde{x})$. Clearly the features that differ between x_0 and $\tilde{x}$ have some influence on the prediction. Hence, non-zero attribution should be given to these features

- Gradient and Gradient $\times$ input fail this axiom
- Consider the following example with just one feature

$$f(x) = 1 - \mathrm{ReLU}(1 - x)$$



Integrated gradients III

- For $x_0 = 0$ and $\tilde{x} = 2$ we obtain $f(x_0) = 0$ and $f(\tilde{x}) = 1$
- However, the gradient at $\tilde{x} = 2$ is zero
- The sensitivity axiom is the complement of the *dummy* property of Shapley values

Integrated gradients IV

Axiom 2: Implementation invariance

Let f and f' be two machine learning models. The two models are *functionally equivalent* if $f(x) = f'(x)$ for all x. Attributions should be identical for functionally equivalent models

- Gradients are invariant to implementations
- Several attribution methods fail this axiom, e.g. LRP and DeepLIFT

Integrated gradients V

Axiom 3: Linearity

Let f be a machine learning model such that

$$f(x) = af_1(x) + bf_2(x)$$

where a and b are weights. The attribution for f is the sum of attributions of f_1 and f_2 weighted by a and b

- Attribution methods should preserve any linearity in the machine learning models

Integrated gradients VI

Axiom 4: Completeness

Let f be a machine learning model. The attribution of all features at a point $\tilde{x}$ should sum up to

$$f(\tilde{x}) - f(x_0)$$

where x_0 is a baseline

- The completeness axiom corresponds to the efficiency property of Shapley values for $f(x_0) = \mathbb{E} f(X)$

Layer-wise relevance propagation (LRP)

Layer-wise relevance propagation (LRP) I

- Layer-wise relevance propagation (LRP) [Bach et al., 2015] exploints the layered structure of neural networks
- Let f be a neural network with L layers
- $R^{(l)} \in \mathbb{R}^{p_l}$ denotes a vector of relevance scores, one for each neuron in the l-th layer
- LRP satisfies the following law of conservation

$$f(x) = \cdots = \sum_{j=1}^{p_{l+1}} R_j^{(l+1)} = \sum_{j=1}^{p_l} R_j^{(l)} = \sum_{j=1}^{p_1} R_j^{(1)}$$

where $R_j^{(1)}$ are the relevances of the input features

Layer-wise relevance propagation (LRP) II

- The output of the neural network f for a given input x is the total relevance, which is distributed among neurons in previous layers
- More specifically, we call any relevance attribution method LRP if it satisfies

$$\begin{aligned} R_i^{(l)} &= \sum_{k:i\to k} R_{i\leftarrow k}^{(l,l+1)} \\ R_k^{(l+1)} &= \sum_{i:i\to k} R_{i\leftarrow k}^{(l,l+1)} \end{aligned}$$

where $R_{i\leftarrow k}^{(l,l+1)}$ is the relevance sent from neuron k to i between layers l and $l+1$

Layer-wise relevance propagation (LRP) III

- The relevances are in both the forward and backward direction sums of the relevances from connecting neurons

- Multiple solutions satisfy these constraints [Montavon et al., 2019], e.g.:
  - Basic rule (LRP-0)

$$R_{i\leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_i w_{ik}}{\sum_j a_j w_{jk}}$$

  - Epsilon rule (LRP-ϵ)

$$R_{i\leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_i w_{ik}}{\epsilon + \sum_j a_j w_{jk}}$$

  - a_i denotes the activation of neuron i (i.e. the output of a neuron before the non-linear activation is applied)

Layer-wise relevance propagation (LRP) IV

- DeepLIFT [Shrikumar et al., 2017] is an extension of LRP that backpropagates relevance values of

$$f(\tilde{x}) - f(x_0)$$

where x_0 is a user defined point that provides a baseline prediction (note that LRP uses $f(x_0) = 0$)

Local interpretable model-agnostic explanations (LIME)

LIME - Basic idea

- Local interpretable model-agnostic explanations (LIME)
- Model-agnostic: We can evaluate the model f but do not make any further assumptions about the model
- In particular, we do not require the model to be differentiable
- LIME locally approximates a machine learning model f using a simple interpretable model g at a specific point $\tilde{x}$ such that

  $$f(\tilde{x}) = g(\tilde{x})$$

  and $f(x) \approx g(x)$ whenever x is close to $\tilde{x}$
- g is typically a linear regression model

LIME - Basic idea

- Given a model class G, we determine a local interpretable model $\hat{g}$ by solving

$$\hat{g} = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_{\tilde{x}}) + \Omega(g)$$

- G could be the class of linear models
- $\mathcal{L}$ is the main loss function we want to minimize
- $\Omega(g)$ is a penalty on the complexity of g, which for instance gives preference to models with fewer parameters
- $\pi_{\tilde{x}}(x)$ is a weight function that measures the proximity of x to $\tilde{x}$, whereby a local approximation of f is enforced

LIME - Basic idea

- Let $x_1, \ldots, x_n$ be a set of n training points
- For regression problems, the $\mathcal{L}$ function can be

$$\mathcal{L}(f, g, \pi_{\tilde{x}}) = \sum_i (f(x_i) - g(x_i))^2 \, \pi_{\tilde{x}}(x_i)$$

- The weight function can be given by an exponential kernel

$$\pi_{\tilde{x}}(x_i) = \exp\left\{ -\frac{d(\tilde{x}, x_i)^2}{\sigma^2} \right\}$$

  where d is a distance function

- σ^2 controls how local the approximation should be

LIME - In practice I

- The data used for training f might not be appropriate for estimating g
- We require samples $(x_i)_i$ close to $\tilde{x}$, where LIME uses the following scheme:
  - Draw a binary vector b_i of length p at random
  - Compute $x_i = h_{\tilde{x}}(b_i)$
- The function $h_{\tilde{x}}$ creates a sample x_i from b_i by masking parts of the reference $\tilde{x}$ (occlusion)

LIME - In practice II

- For instance, $h_{\tilde{x}}$ could mask features by replacing them with feature means $\bar{x}^{(j)}$, i.e. the jth feature of $x_i = h_{\tilde{x}}(b_i)$ is given by

$$x_i^{(j)} = \begin{cases} \tilde{x}^{(j)} & \text{if } b_i^{(j)} = 1 \\ \bar{x}^{(j)} & \text{if } b_i^{(j)} = 0 \end{cases}$$

LIME - In practice III

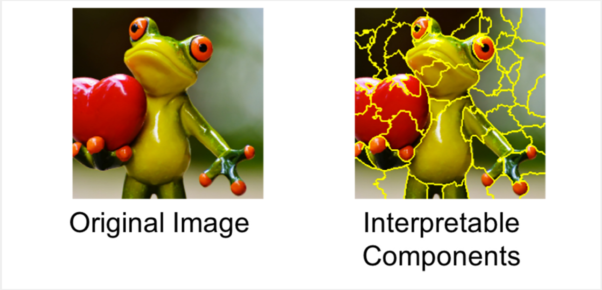
- Example topic models:
  - A data point x is a word count vector, where each entry $x^{(j)}$ denotes the number of occurrences of word j in a text document
  - $h_{\tilde{x}}(b_i)$ could mask features by replacing them with zeros, i.e.

$$h_{\tilde{x}}(b_i) = \begin{cases} \tilde{x}^{(j)} & \text{if } b_i^{(j)} = 1 \\ 0 & \text{if } b_i^{(j)} = 0 \end{cases}$$

  - Hence, samples x_i are created from the reference $\tilde{x}$ by replacing some of the counts with zeros

LIME - In practice IV

- Example image classification:
  - x is a an image, where each entry $x^{(j)}$ denotes a pixel or super-pixel
  - A sample $x_i = h_{\tilde{x}}(b_i)$ consists of the reference image $\tilde{x}$ where a some pixels or super-pixels have been masked, as defined by the binary vector b_i



LIME - In practice V

- The interpretable model g is typically defined on the binarized points b_i
- The loss for regression problems then becomes

$$\mathcal{L}(f, g, \pi_{\tilde{x}}) = \sum_i (f(x_i) - g(b_i))^2 \, \pi_{\tilde{x}}(x_i)$$

Shapley values

Shapley values - Motivation

- Assume a linear model

$$f(x) = \theta_1 x^{(1)} + \theta_2 x^{(2)} + \cdots + \theta_p x^{(p)}$$

- If features are standardized we can interpret the coefficient θ_j as the *global* importance of the *j*th feature

- Given a specific input x, the contribution of feature j to the prediction $f(x)$ is given by

$$\begin{aligned} \phi_j(f, x) &= \theta_j x^{(j)} - \mathbb{E}\left[\theta_j X^{(j)}\right] \\ &= \theta_j \left(x^{(j)} - \mathbb{E}\left[X^{(j)}\right]\right) \end{aligned}$$

assuming features are independent

Shapley values - Motivation I

- For non-linear models we need a more advanced definition
- Let F denote the set of p features and $S \subseteq F$ a subset
- Furthermore, let $f(x^{(S)})$ be the prediction of a machine learning model where only a subset of features S is used
- Let $S = F \backslash \{j\}$, then the contribution of the jth feature can be measured as

$$f(x^{(S \cup \{j\})}) - f(x^{(S)})$$

  - $f(x^{(S \cup \{j\})})$ is the prediction with feature j
  - $f(x^{(S)})$ the prediction without feature j

Shapley values - Motivation II

- In practice, features are rarely independent, i.e. feature j might only be informative in combination with other features
- In this case we have to attribute some of feature jth contribution to those features
- We have to test for all subsets $S \subseteq F \setminus \{j\}$

Shapley values - Example I

- Let the feature set F consist of $p = 3$ elements, i.e. $F = \{1, 2, 3\}$

- Assume we observe the following predictions

$$
\begin{aligned}
&f(x^{(\{1\})}) = 100, \quad f(x^{(\{1,2\})}) = 500, \\
&f(x^{(\{2\})}) = 100, \quad f(x^{(\{1,3\})}) = 300, \quad f(x^{(\{1,2,3\})}) = 1100 \\
&f(x^{(\{3\})}) = 100, \quad f(x^{(\{2,3\})}) = 300,
\end{aligned}
$$

- Clearly, features are not contributing independently to the predictions

- For independent features we would expect

$$f(x^{(\{1,2\})}) = f(x^{(\{1\})}) + f(x^{(\{2\})})$$

Shapley values - Example II

- How much should we attribute to each feature?
- We fix a particular feature j and evaluate its contribution to all subsets $S \subseteq F \setminus \{j\}$
- To simplify notation, let

$$\xi_j(S) = f(x^{(S \cup \{j\})}) - f(x^{(S)})$$

- For $j = 3$ and $S = \{1, 2\}$ we have

$$\xi_j(S) = 1100 - 500 = 600$$

- For $j = 2$ and $S = \{3\}$ we have

$$\xi_j(S) = 300 - 100 = 200$$

Shapley values - Example III

- $\xi_j(S)$ denotes the contribution of feature j to the prediction based on features S
- The Shapley value for feature j is the average over all contributions
- We evaluate all $p!$ permutations of p features, i.e.

$$\begin{gathered} 1, 2, 3 \\ 1, 3, 2 \\ 2, 1, 3 \\ 2, 3, 1 \\ 3, 1, 2 \\ 3, 2, 1 \end{gathered}$$

Shapley values - Example IV

- A permutation is interpreted as a sequence of features entering the set of features S
- For instance, for $2, 1, 3$ we first have feature 2 entering S and afterwards feature 1. Feature 3 is the last to join S
- We then evaluate the contribution of each feature, i.e. for $2, 1, 3$ we evaluate $\xi_2(\{\})$, $\xi_1(\{2\})$, and $\xi_3(\{1, 2\})$

Shapley values - Example V

	$j=1$		$j=2$		$j=3$	
1,2,3	$\xi_1(\{\})$	$=100$	$\xi_2(\{1\})$	$=400$	$\xi_3(\{1,2\})$	$=600$
1,3,2	$\xi_1(\{\})$	$=100$	$\xi_2(\{1,3\})$	$=800$	$\xi_3(\{1\})$	$=200$
2,1,3	$\xi_1(\{2\})$	$=400$	$\xi_2(\{\})$	$=100$	$\xi_3(\{1,2\})$	$=600$
2,3,1	$\xi_1(\{2,3\})$	$=800$	$\xi_2(\{\})$	$=100$	$\xi_3(\{2\})$	$=200$
3,1,2	$\xi_1(\{3\})$	$=200$	$\xi_2(\{1,3\})$	$=800$	$\xi_3(\{\})$	$=100$
3,2,1	$\xi_1(\{2,3\})$	$=800$	$\xi_2(\{3\})$	$=200$	$\xi_3(\{\})$	$=100$

- The rows are the permutations, the columns represent features to enter the set S
- The Shapley value $\phi_j(f,x)$ for feature j is the average over all $p! = |F|!$ rows in column j
- Hence, permutations are assumed to be uniformly distributed

Shapley values - Example VI

- How often do we observe a particular entry $\xi_j(S)$ in column j?
- We can permute all features before j enters and all features after j enters
- Hence, an entry $\xi_j(S)$ occurs

$$|S|!(|F| - |S| - 1)!$$

times in column j

Shapley values - Definition I

Shapley value [Shapley, 1951]

The shapley value for the *j*th feature is defined as

$$\phi_j(f,x) = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \xi_j(S)$$
$$= \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left(f(x^{(S \cup \{j\})}) - f(x^{(S)})\right)$$

- The sum is over 2^{p-1} permutations
- For large feature sets the Shapley value is computationally very expensive or even impossible to compute

Shapley values - Linear models I

- Assume f is a linear model of the from

$$f(x) = \theta_1 x^{(1)} + \theta_2 x^{(2)} + \cdots + \theta_p x^{(p)}$$

- Given independent features, the Shapley values for this model reduce to

$$\phi_j(f, x) = \theta_j \left(x^{(j)} - \mathbb{E}\left[X^{(j)} \right] \right)$$

  [Štrumbelj and Kononenko, 2014]

- This is what we expected from our previous discussion

Shapley values - Properties I

- *Efficiency:*
$$\sum_j \phi_j(f,x) = f(x) - \mathbb{E}_X f(X)$$

- *Symmetry:* If two features j and k contribute equally to all subsets, then
$$\phi_j(f,x) = \phi_k(f,x)$$
for all x

- *Dummy:* If feature j does not influence the prediction $f(x^{(S)})$ for all S, then
$$\phi_j(f,x) = 0$$

Shapley values - Properties II

- *Additivity:* If $f(x) = \sum_m f_m(x)$ then

$$\phi_j(f, x) = \sum_m \phi_j(f_m, x)$$

  i.e. f could be a random forest or any other bagging method

Shapley values - In practice I

- How do we remove features from the prediction of our machine learning model f?
- The optimal but impractical way would be to train a model f_S for each subset S
- Instead, we often use

$$f(x^{(S)}) = \mathbb{E}\left[f(X)\,|\,X^{(S)} = x^{(S)}\right]$$

  where all elements of X that are not given by $\{X^{(S)} = x^{(S)}\}$ are considered random

- The expectation can be estimated from our training data, which however requires many evaluations of the model f

Shapley values - In practice II

- Assuming that our model f is linear, we obtain

$$f(x^{(S)}) = \mathbb{E}\left[f(X) \mid X^{(S)} = x^{(S)}\right] = f\left(\mathbb{E}[X \mid X^{(S)} = x^{(S)}]\right)$$

- Furthermore, assuming independent features we obtain

$$f(x^{(S)}) = f(\bar{x}^{(S)})$$

where

$$\bar{x}^{(S)} = \begin{cases} x^{(j)} & \text{if } j \in S \\ \mathbb{E}\, X^{(j)} & \text{if } j \notin S \end{cases}$$

i.e. all features not in S have been replaced by their expectation

Shapley values - Monte Carlo I

- Summing over 2^{p-1} contributions is often too expensive
- We may utilize Monte Carlo approximations (law of large numbers) to estimate the Shapley value [Štrumbelj and Kononenko, 2014]
  - Draw k permutation $\pi_i = (r_1, \ldots, r_p)$ with $r_m \in \{1, \ldots, p\}$ from a uniform distribution
  - For each permutation π_i, compute the set of features S_{ij} from π_i, i.e. all features until feature j occurs in π_i
  - The Monte Carlo approximation of the Shapley value is given by

$$\phi_j(f, x) \approx \frac{1}{k} \sum_{i=1}^{k} \left(f(x^{(S_{ij} \cup \{j\})}) - f(x^{(S_{ij})}) \right)$$

Shapley values - Kernel SHAP I

- SHapley Additive exPlanations (SHAP) [Lundberg and Lee, 2017]
- Kernel SHAP reformulates the computation of Shapley values as a linear regression problem using the LIME framework
- The interpretable model g is assumed to be a linear regression model

$$g(b_i) = \phi_0 + \sum_{j=1}^{p} \phi_j b_i^{(j)}$$

i.e. the contributions of the linear model depend on the weights ϕ_j and the binary values $b_i^{(j)}$

Shapley values - Kernel SHAP II

- The weights ϕ_j are the Shapley values
- Notice that LIME with loss function

$$\mathcal{L}(f, g, \pi_{\tilde{x}}) = \sum_i \left(f(x_i) - g(b_i)\right)^2 \pi_{\tilde{x}}(x_i)$$

and $\Omega(g) = 0$ corresponds to weighted ordinary least squares

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta} \left\| W^{1/2}(y - X\theta) \right\|_2^2 \\ &= (X^\top W X)^{-1} X^\top W y\end{aligned}$$

where $X \in \{0, 1\}^{2^p \times p}$ denotes a matrix containing all possible binary vectors b_i of length p as rows, $W = (w_{ii})$ is a weight

Shapley values - Kernel SHAP III

matrix with $w_{ii} = \pi_{\tilde{x}}(x_i)$ and $y = (y_i)$ is the vector of targets $y_i = f(x_i)$

- The coefficients $\hat{\theta}$ are the Shapley values $\phi = (\phi_1, \dots, \phi_p)$ for

$$\pi_{\tilde{x}}(x_i) = \frac{p-1}{\binom{p}{k_i} k_i (p-k_i)}$$

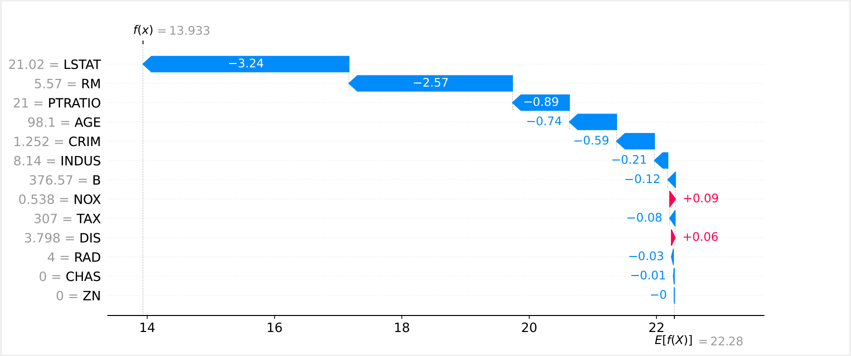
where $k_i = |b_i|$ is the number of ones in the binary representation of the ith sample

- Notice that $|b_i|$ measures the similarity between x_i and x, therefore this particular choice of $\pi_{\tilde{x}}$ is indeed a weight based on a distance measured

Shapley values - Kernel SHAP IV

- Recall that the *i*th sample x_i is generated from x by randomly generating a binary representation b_i and afterwards masking all features j in $\tilde{x}$ where $b_i^{(j)} = 0$
- The linear regression coefficients θ correspond to the Shapley values ϕ only when we consider all possible binary vectors b_i
- In practice, Kernel SHAP uses a sampled subset of binary vectors
- An improved method has been proposed [Kwon and Zou, 2022]

Shapley values - Example



Partial Dependence Plot

Partial Dependence Plot

- Let f be a black-box model such as a neural network
- What is the effect of individual predictors $X^{(j)}$ on the response variable Y as captured by our model f?

Partial Dependence Plot

- Partial dependence plots (PDP) [Friedman, 2001]:

$$\mathrm{PDP}_j(x) = \int f(x, x^{(-j)}) \mathrm{pr}(x^{(-j)}) dx^{(-j)}$$

  where $x^{(-j)} = (x^{(1)}, \ldots, x^{(j-1)}, x^{(j+1)}, \ldots x^{(p)})$

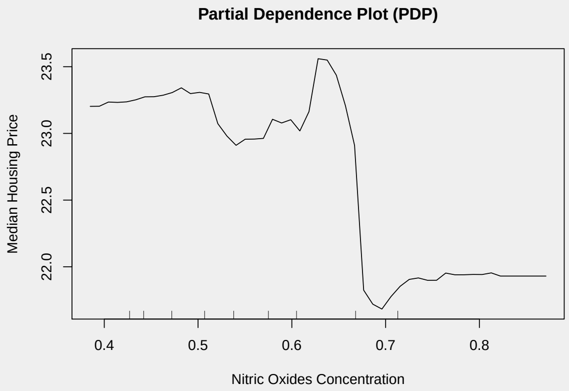
- In practice we use the training data $(x_i, y_i)_{i=1}^n$ to estimate the PDP, i.e.

$$\widehat{\mathrm{PDP}}_j(x) = \frac{1}{n} \sum_{i=1}^{n} f(x, x_i^{(-j)})$$

Partial Dependence Plot - Example

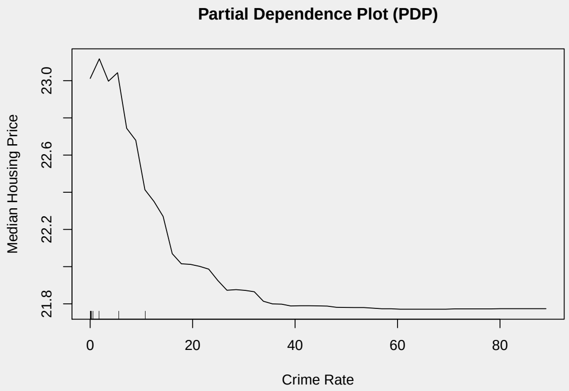
- Boston housing data: Housing data for 506 census tracts of Boston from the 1970 census
- X: capita crime rate, proportion of non-retail business acres per town, nitric oxides concentration, average number of rooms per dwelling, proportion of owner-occupied units built prior to 1940, ...
- Y: median value of owner-occupied homes in USD 1000's

Partial Dependence Plot - Example



- Housing prices drop when *nitric oxides concentration* reaches $\sim$ 0.68

Partial Dependence Plot - Example



- Housing prices drop quickly with crime rate

Partial Dependence Plot - ICE

- The individual conditional expectation (ICE) is an extension of the PDP
- It plots each component of the PDP sum individually, i.e.

$$\widehat{\mathrm{ICE}}_{ij}(x) = f(x, x_i^{(-j)})$$

- Hence, we have

$$\widehat{\mathrm{PDP}}_j(x) = \frac{1}{n} \sum_{i=1}^{n} \widehat{\mathrm{ICE}}_{ij}(x)$$

Input Optimization

Input optimization I

- Assume that f is a classifier for images
- We want to find inputs x not contained in the training set that correspond to predictions of a given classification
- This analysis might help to understand if f is sensitive to the correct features
- For a given output y we solve the optimization problem

$$\hat{x} = \arg\min_{x} \mathcal{L}(f(x), y)$$

- The loss function $\mathcal{L}$ typically corresponds to the loss function used for training f

Input optimization II

- As for training f we may use gradient descent to compute $\hat{x}$
- The result $\hat{x}$ depends strongly on the initial value for solving the optimization problem
- Using multiple initial conditions allows to generate multiple inputs $(x_i)_i$ corresponding to the same prediction y

SOFTWARE

- SHAP:
  `https://shap.readthedocs.io`
- iNNvestigate (Keras/Tensorflow):
  `https://github.com/albermax/innvestigate`
- Captum (PyTorch):
  `https://github.com/pytorch/captum`

References I

Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. (2017).
Towards better understanding of gradient-based attribution methods for deep neural networks.
arXiv preprint arXiv:1711.06104.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015).
On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation.
PloS one, 10(7):e0130140.

Breiman, L. (2001).
Statistical modeling: The two cultures (with comments and a rejoinder by the author).
Statistical science, 16(3):199–231.

Friedman, J. H. (2001).
Greedy function approximation: a gradient boosting machine.
Annals of statistics, pages 1189–1232.

References II

Kim, B., Seo, J., Jeon, S., Koo, J., Choe, J., and Jeon, T. (2019).
Why are saliency maps noisy? cause of and solution to noisy saliency maps.
In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 4149–4157. IEEE.

Kwon, Y. and Zou, J. (2022).
Weightedshap: analyzing and improving shapley based feature attributions.
arXiv preprint arXiv:2209.13429.

Lundberg, S. M. and Lee, S.-I. (2017).
A unified approach to interpreting model predictions.
Advances in neural information processing systems, 30.

References III

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., and Müller, K.-R. (2019).
Layer-wise relevance propagation: an overview.
Explainable AI: interpreting, explaining and visualizing deep learning, pages 193–209.

Shapley, L. S. (1951).
Notes on the n-person game—ii: The value of an n-person game.(1951).
U.S. Airforce PROJECT RAND - Research Memorandum.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017).
Learning important features through propagating activation differences.
In *International conference on machine learning*, pages 3145–3153. PMLR.

References IV

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. (2016).
Not just a black box: Learning important features through propagating activation differences.
arXiv preprint arXiv:1605.01713.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013).
Deep inside convolutional networks: Visualising image classification models and saliency maps.
arXiv preprint arXiv:1312.6034.

Štrumbelj, E. and Kononenko, I. (2014).
Explaining prediction models and individual predictions with feature contributions.
Knowledge and information systems, 41(3):647–665.

References V

Sundararajan, M., Taly, A., and Yan, Q. (2017).
Axiomatic attribution for deep networks.
In *International conference on machine learning*, pages 3319–3328. PMLR.

Zhao, Q. and Hastie, T. (2021).
Causal interpretations of black-box models.
Journal of Business & Economic Statistics, 39(1):272–281.