FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Machine Learning Explainability Methods
Student:	Danila Makulov
Supervisor:	Mgr. Vojtěch Rybář
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Machine learning models are increasingly being used in many applications, but they can be difficult to understand and interpret. Explainability is becoming an increasingly important factor in the selection of machine learning models. This thesis aims to review and evaluate various explainability methods for machine learning models.

1. Review and summarise the current state of explainability methods for machine learning models. Focus on methods for tabular data and neural networks.
2. Evaluate behaviour of at least three explainability methods for models on tabular data(e.g. Individual Conditional Expectations, SHAP, LIME...) and two for neural networks (Saliency Maps, Influential Instances...).
3. Try to find examples where various methods contradict each other.
4. Analyse the impact of correlated data on explainability results.
5. Based on both literature review and own experiments propose conditions or heuristics where explainable methods outputs are consistent.

Reference:

1. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, https://christophm.github.io/interpretable-ml-book/.
2. Adebayo, J., Muelly, M., Abelson, H. and Kim, B., 2022. Post hoc explanations may be ineffective for detecting unknown spurious correlation. In International Conference on Learning Representations.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 30 January 2023 in Prague.

Bachelor's thesis

MACHINE LEARNING EXPLAINABILITY METHODS

Danila Makulov

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Vojtěch Rybář
January 11, 2024

Contents

List of Figures

List of Tables

I would like to express my deepest gratitude to my supervisor, Vojtěch Rybář, for all the support and help while writing this work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on January 11, 2024 .....

Abstract

Machine Learning is becoming more and more used in many sensitive applications where it is essential to understand why the models behave as they do. Such a rapid increase has heightened the demand for Explainable Machine Learning and new explanation methods. These methods, however, are not guaranteed to yield consistent outputs.

This work give a concise overview of the current state of Explainable Machine Learning and its methods, focusing primarily on the local explanation methods (e.g. SHAP and LIME) and global plotting methods for tabular data, and methods specific to neural network models. We show examples of inconsistent explanations of SHAP and LIME, illustrate and explain how some methods are impacted by correlation, and show practical examples of using neural network methods to analyze the model and find its biases. In the end we give some recommendations when dealing with inconsistent outputs based on the research we made and our own experiments.

Keywords Machine Learning, Machine Learning Interpretability, xAI, Correlation, Disagreement Problem, SHAP, LIME, PDP, PFI, Saliency Maps, Influential Functions

Abstrakt

Strojové učení se stále více používá v mnoha citlivých aplikacích, kde je nezbytné pochopit, proč se modely chovají tak, jak se chovají. Takový rychlý nárůst zvýšil poptávku po vysvětlitelném strojovém učení a nových metodách vysvětlení. Tyto metody však nezaručují konzistentní výstupy.

Tato práce podává stručný přehled současného stavu vysvětlitelného strojového učení a jeho metod se zaměřením především na metody lokálního vysvětlení (např. SHAP a LIME) a globální metody vykreslování pro tabulková data a metody specifické pro modely neuronových sítí. Ukazujeme příklady nekonzistentních vysvětlení SHAP a LIME, ilustrujeme a vysvětlujeme, jak jsou některé metody ovlivněny korelací, a ukazujeme praktické příklady použití metod pro neuronové sítě k analýze modelu a nalezení jeho biasů. Na závěr uvádíme několik doporučení při řešení nekonzistentních výstupů na základě provedeného výzkumu a vlastních experimentů.

Klíčová slova Strojové učení, Vysvětlitelnost, xAI, Korelace, Disagreement Problem, SHAP, LIME, PDP, PFI, Saliency Maps, Influential Functions

Abbreviation List

ML	Machine Learning
AI	Artifical Intelligence
xAI	eXplainable Artifical Intelligence
SHAP	SHapley Additive exPlanations
LIME	Local Interpretable Model-Agnostic Explanations
PD	Partial Dependence
PDP	Partial Dependence Plot
M Plot	Marginal Plot
ALE	Accumulated Local Effects
PFI	Permutation Feature Importance
FFNN	Feed Forward Neural Network
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
RMSE	Root-Mean-Square Error

Chapter 1

Introduction

1.1 Motivation

In the current world algorithms and machine learning models are being used more and more in the sensitive areas. Example of a COMPAS model - a recidivism risk-scoring model used throughout the criminal justice system in the U.S. - shows how the model can be unfair towards some groups of people, for example through defendant's age [1]. Understanding why this happens and what we can do about it is crucial and it is the task of explainable machine learning. This work aims to give an overview of the current state of explainability in machine learning and from the practical point of view use it to analyze possible contradictions between different methods and see how single methods deal with the problems in machine learning such as correlation in data.

1.2 Background

Machine Learning (ML) is becoming more and more used technique for solving complex tasks. Machine Learning includes a lot of models, but there is no model universally good for all of the tasks: some are better for some tasks, others are better for the others. Complex models, such as deep neural networks, are much more suitable for the big data tasks. These complex models work as a "black-box": they get an input in the form of tabular data, text or images and produce the output, but there is no easy way to understand why this particular output was made. Other black-box models include number of ensemble methods: models consisting of numerous "simpler" ML models, which are also widely used in many different areas.

The opposite of the black-box models are, unsurprisingly, white-box models. In the field of xAI they are also called "Interpretable Models". In addition to producing output, they also give an interpretation (or explanation) to it, which is their benefit as opposed to the black-box models.

Because of the rapid increase in deployment of complex black-box models, their explanation is becoming more and more important area of AI and there has been a number of works done in this field: Christoph Molnar in his book "Interpretable Machine Learning" gives very good overview of existing explainabity methods [2]; different researches analyzed how problematic data, such as correlated data, might affect the results, for example, by producing controversial and erroneous explanations [3]; with increasing demand in the area, more explainability methods are being designed, all using different approaches, which may lead to controversial explanations between single methods, rather new problem known as "Disagreement Problem" [4].

1.3 Aims

This work is mainly divided into 2 parts.

The theoretical part aims to give an overview of the current state of the machine learning explainability methods, describe how and why they are affected by correlated data and take a brief look into the ”disagreement problem”: when different methods give different, controversial results [4].

In the practical part we will be aiming to evaluate methods reviewed in the theoretical part on the tabular data and images, see the correlation impact and contradictions in methods on practice. At the end, we will also be giving some recommendations on how to deal with contradicting explanations based on our literature review and results from the practical part.

1.4 Limitations

This work will not be giving an in-depth explainability methods overview because of their high number and for the same reason will not review all of them. Also, the work will be focusing primarily on the methods working with tabular data and two methods specific to neural networks.

1.5 Work Structure

1. Literature review part, aiming to give an overview of different kinds of explanation methods, review them, analyze the impact of correlation on them and review the disagreement problem.
2. Practical part, aiming to use and evaluate methods from the literature part in practice on tabular data, show the impact of correlation, compare different pairs of methods to see how their results are inconsistent. It will also be giving practical examples of two different methods specific to neural network models and how they can be used to debug the model and find biases.
3. Recommendation chapter where we will be giving recommendations on how to deal with disagreements in the explanations when data is correlated and/or when selecting between different explanation methods.
4. Conclusion.

Chapter 2
Literature Review

2.1 What is Explainable Machine Learning?

Explainable Machine Learning or Explainable Artificial Intelligence (xAI) is a highly promising and very rapidly developing area of Machine Learning and not without a reason. Generally, the difference between regular Machine Learning and Explainable one is that the latter in addition to making a prediction provides an explanation to it. [1]

Let us first consider why we might need to explain predictions in Machine Learning.

2.2 Reasons for xAI

Explaining black-boxes is becoming more important task in the recent years and here is why:

1. Explanations help people build trust in the underlying ML models: instead of blindly believing that the model or an algorithm works properly, one would prefer to understand how it works.
2. It is argued that some laws already enforce the widely-used algorithms[2] to provide an explanation for their solutions or such laws can come later [5].
3. Explanations help to debug models. Both to understand why the incorrect and correct prediction was made which is useful to check whether the model is biased and why.
4. Explanations help to better understand the model and what it learnt.

Note: the list is far from being full and, as both xAI and AI go on, the list is sure to be updated with new reasons.

2.3 What is an explanation?

One way to define an explanation is as a process of clarifying and making understandable the predictions made by the machine learning model or the model itself. Throughout this work we will be viewing an explanation as an output of the explainability method. It is worth noting

[1] Note: terms "explainability" and "interpretability" will be used interchangeably throughout all the work, similar way they are used in the xAI field.

[2] In today's world term "algorithm(s)" might be often used instead of "ML model" even though they are distinct: usually, ML model is not an algorithm. e.g. "recommendation algorithms" in different services and applications)

that sometimes one single method can output a number of different explanations, e.g. SHAP [6]. Christoph Molnar in his book that overviews the current state of the interpretable AI [2] specifies the following explanations:

- Feature summary statistic. Mostly for tabular data, the output of such explanation method is some number per feature describing how single feature affects the model's prediction locally (for one single prediction) or globally. Examples are SHAP [6] or LIME [7] methods which output explanations in the form of *feature importances*.
- Feature summary visualization. When feature effects are visualized. Example could be Partial Dependence Plot [8] which plots marginal dependence of a feature upon the target value, or Pixel Attribution methods that highlight pixels that were the most relevant for prediction of a neural network model.
- Data points. The output of these methods is a set of different data points. For example, Influential Instances method can find the most important training instances for the model.

2.4 Taxonomy of explainability methods

Explainable Artificial Intelligence is a vast area including many different explainability methods, all used for different goals, for different users - such as practitioners and people not much familiar with intrinsics of ML - and calculated in different ways.

In this chapter we will be giving an overview of post-hoc explanation methods in terms of the following: scope and model-specificity [2].

In terms of scope, explanation methods could be either *local* or *global*. Local methods explain single prediction made by the model, while global explain the model itself.

In terms of model-specificity, explanation methods could be *model-agnostic* or *model-specific*. Model-specific methods are applicable only to the certain type of models, because they use some properties intrinsic to that type. For example, TreeSHAP [6] is used for finding feature attributions for the tree-based models, or Saliency Maps [9] make use of the internal structure of neural networks used with image data highlighting image pixels that the model used the most. In turn, model-agnostic methods, as the name tells, can be used with any model, which is their strength, because of today's ML modularity - underlying models can be changed - and because one can use one method in multiple different applications. These methods treat any model as a black-box, making use only of its prediction function [2].

Before we start our overview, let us first see how some ML models can actually be interpretable.

2.5 Interpretable Machine Learning Models

Interpretable Machine Learning models are models that can be interpreted/explained directly without using any additional explainability methods. In the context of xAI, they might also be called "white-box" models, as opposed to "black-box" ones. As a rule, these models are quite simple and not as complex as black-boxes. They include linear regression models and decision trees.

2.5.1 Linear Regression

Of course, the list is not full and when we say linear regression we mean all the regression models with different loss functions, e.g. ordinary MSE with L1 or L2 regularization terms (Lasso and Ridge regressions, respectively) since cost function that is being optimized does not play a huge role.

What plays role in the interpretability of linear regression, however, are the feature weights. Given p independent features $X_1, \ldots, X_p$ and dependent (target) variable Y, linear regression assumes following relationship between them:

$$Y = \sum_{i=1}^{p} w_i X_i + w_0 + \epsilon$$

where w_i is the feature weight for variable X_i, w_0 is the intercept and ϵ is the error term that follows normal distribution with mean 0. Then it finds the weights $\hat{w}_i$ for every X_i by finding the optimal solution of the cost function - usually Mean Squared Error - across n observations.

In xAI these feature weights w_i can be considered a form of explanations. Given an instance of interest x with values $x_1, \ldots, x_p$ we can find the feature that contributed to its prediction the most by multiplying each feature value by the appropriate weight $\hat{w}_i$ and getting the maximum: $\max(\{\hat{w}_1 x_1, \ldots, \hat{w}_p x_p\})$. The same goes when analyzing the model, without any instances. Looking at the weights by themselves, we can see which feature contributes to the model prediction the most by finding the highest feature weight. Here, however, the caveat is that since X variables do not follow the same distribution, comparing only the weights could be, and probably is, meaningless without any data preprocessing beforehand. For example, imagine that we have 2 same variables X_1 and X_2 measuring the distance, but X_1 measures it in meters, while X_2 in centimeters. Now imagine also that linear regression model gives these features the weights 1 and 100. Comparing only the weight values, one would draw a conclusion that X_1 must have more influence on the model since it has higher feature weight. This, however, is not true because with these weights feature effects on a prediction of a single instance would be the same. This problem can be easily avoided by standardizing the data before training the model as shown in [2]. Once the data is standardized (mean is zero and variance is one), we can compare the feature weights alone, and feature weights can be said to be *feature importances*.

2.5.2 Decision Trees

Decision trees are even more intuitive in their interpretability. Consider decision trees used for binary classification task. Decision tree builds a recursive tree of a given depth by trying to split the observed data into 2 (or more) groups/nodes so that each group has as many same classes as possible by finding a decision rule that maximizes Information Gain function. For example, given an observation with classes $\{1, 1, 1, 1, 1, 0, 0, 0\}$, the goal is to find a decision rule that splits it into $\{1, 1, 1, 1, 1\}$ and $\{0, 0, 0\}$. The prediction in the single node is then the average class of all the classes in the observation of the node. The prediction for a new instance is then the prediction in the terminal node of the decision tree.

To explain the prediction of the single instance with the decision tree, we should follow the decision path of this instance. The output is the ordered list of the rules that had the most effect on this prediction ordered from the most important (decision rule of the root node) to the least (terminal node).

The same applies to the global analysis: the decision rules from the upper nodes in the tree are more important for the model than those in the lower ones.

2.6 Global Model-Agnostic Methods

Global Model-Agnostic explainability methods can be used to explain practically any model. Global methods aim to evaluate how model behaves on average.

Global methods usually generate some form of explanation for one feature or a group of them. It could be a plot describing the relationship between a given feature and a target variable or a bar plot showing the most important features for the model (*feature importance*). It is important to note, that in all cases, we want to see the relationships that the model learnt from

the (training) data, not the real relationships of the data. Of course, ideally, we want to have a model that can perfectly describe data distribution and therefore make use of it in the future predictions, but it could be difficult (or in some cases even impossible) to do so; moreover, we might simply have a bad model that overfits or underfits in which case the model definitely has bad understanding of the data on which it was trained. In any case, we are trying to understand some sort of relation between the model and the data, not the data directly.

Let us first take a look at the plot methods: methods that as an explanation generate a plot showing the dependence of one or two features and the predicted variable.

2.6.1 Partial Dependence Plot

Partial Dependence Plot (PDP) [8] visualizes marginal effect a feature (or group of features) have on the target variable. It shows whether the relationship between a feature and a target variable is linear, monotonic or more complex. Figure 2.1 shows the example for PDP. The formula for the partial dependence function $f_{S,PDP}$ is

$$f_{S,PDP}(x_S) = E[f(x_S, X_C)]$$

The S is the subset of features we want to plot our function for, x_S are the values for this subset. Other features $C = X \setminus S$ are treated as random variables.

To estimate $f_{S,PDP}$ we can use the following formula:

$$\hat{f}_{S,PDP}(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_S, x_C^i) \tag{2.1}$$

Usually, number of features in S is 1 or 2, since it's more difficult to visualize plots with 3 variables. For simplicity, let us consider that S includes only one feature. In the formula 2.1 partial dependence function for this feature for the given value s would be calculated as the mean value of predicted values of our model if we substitute every value of feature S with the value s while all the other values stay the same. This gives us an intuitive interpretation of this function: its value represents the average prediction if we force all data points to have that feature value [2].

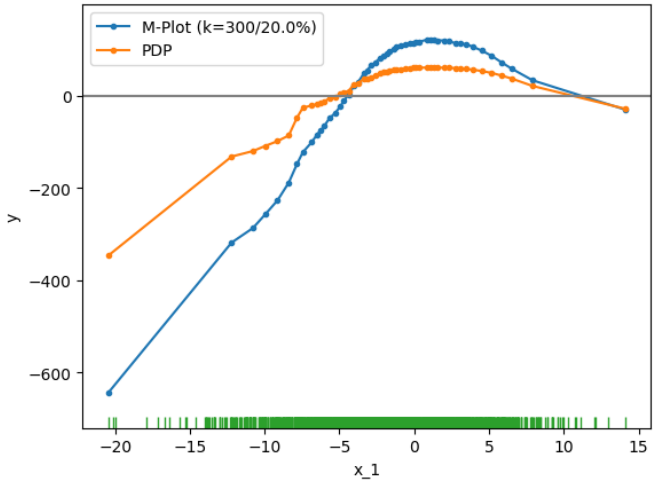


Figure 2.1 Example of PDP plot (centered around average prediction) from the practical part. We can see quadratic relationship between feature x_1 and target variable.

Important assumption here is that features in S are not correlated with other features in C: if they are, partial dependence will average on unrealistic data instances on which the model hasn't been trained, leading to incorrect estimations. This issue is caused by marginal distribution of a given feature meaning that it is assumed that it does not depend on other features, which can be solved by using conditional distributions which is exactly what ALE and M plots do - methods we will be reviewing next.

It is worth noting that since PDP shows the relationship that the model has learnt for the given feature, for linear regression model PDP shows a straight line because of its linearity.

In other words, if we consider a linear regression model with weights $w_1, ..., w_p$ and intercept w_0, then the derivation of PDP function with respect to x_s should be w_S.

The prediction function for a linear model $\hat{f}$ at x^i is:

$$\hat{f}(x^i) = w_1 x_1^i + ... + w_p x_p^i + w_0$$

Then, as we already said, if we take partial derivative of PDP with respect to x_s:

$$\frac{d\hat{f}_{S,PDP}}{dx_s} = \frac{d(\frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_s, x_C^i))}{dx_s} = \frac{1}{n}\frac{d(\sum_{i=1}^{n}(w_1 x_1^i + ... + w_S x_s + ... + w_p x_p^i + w_0))}{dx_s} = n\frac{1}{n}\frac{d(w_S x_s)}{dx_s} = w_S$$

we get a weight of the feature S: w_S. This means that the slope of PDP function for any feature of a linear regression model is always constant and, therefore, the relationship between this feature and a target variable is linear. This is trivial, yet useful, observation we will use in the practical part while comparing explanations produced by different methods on a linear model.

PDP plots can be computed with a number of python packages, such as `scikit-learn` [10] or `alepython`[3].

2.6.2 Marginal Plot

From the PDP section we know that partial dependence function might create unrealistic data points if features are correlated. Marginal Plot (M plot) [11] addresses this issue by using conditional distributions. For two features M Plot of the effect X_1 is a plot of the function

$$f_{1,M}(x_1) = \mathbb{E}[f(X_1, X_2)|X_1 = x_1] = \int p_{2|1}(x_2|x_1) f(x_1, x_2) dx_2$$

We can estimate $f_{1,M}(x_1)$ as

$$\hat{f}_{1,m}(x_1) = \frac{1}{n(x_1)} \sum_{i \in N(x_1)} f(x_1, x_{i,2})$$

where $N(x_1)$ is sufficiently small neighborhood of x_1, and $n(x_1) = |N(x_1)|$.

We can see the difference in the PDP and M Plot function computations in Figure 2.2.

But M plot comes with another problem. Because of its use of feature's conditional distribution, feature's effect becomes entangled with the effects of features it is correlated with. So even though M plot does not generate feature's effect over improbable data instances, it can mix the effects of different correlated features into one, making the interpretation of resultant plot more difficult. However, looking ahead, as we will show in the section overviewing the correlation impact in xAI, mixing of features effects is not necessarily harmful, and could even be desirable. But in the meantime, there is another plotting method that can deal with both marginal distribution and mixing correlated features together.

[3] `https://github.com/blent-ai/ALEPython`

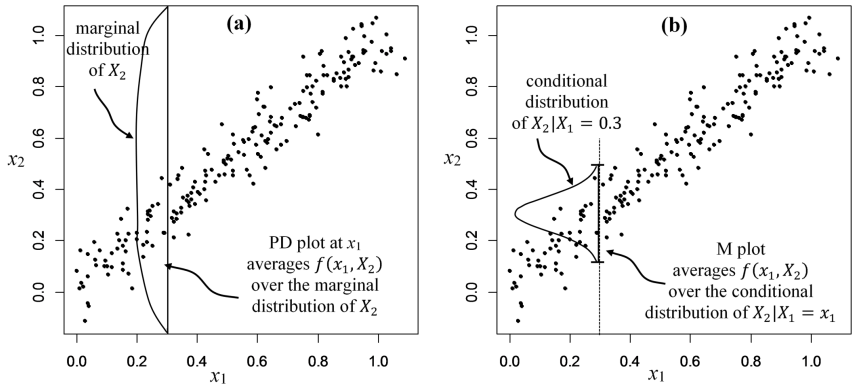


Figure 2.2 Illustration of the differences between the computation of (a) $f_{1,PD}(x_1)$ and (b) $f_{1,M}(x_1)$ at $x_1 = 0.3$. Taken from [11].

2.6.3 Accumulated Local Effects

Accumulated Local Effects (ALE) [11] is another plot method that extends the idea of Marginal Plot to utilize feature's conditional distribution instead of marginal one, but it also addresses the issue when the resultant plot mixes the effect of the analyzed feature with ones correlated with it.

First and foremost, for two features X_1 and X_2 authors define the ALE main-effect of X_1 as

$$f_{1,\mathrm{ALE}}(x_1) = \int_{x_{\min,1}}^{x_1} \mathbb{E}[f^1(X_1, X_2)|X_1 = z_1]dz_1 - \text{constant}$$
$$= \int_{x_{\min,1}}^{x_1} \int p_{2|1}(x_2|z_1) f^1(z_1, x_2) dx_2 dz_1 - \text{constant}$$

where $f^1(x_1, x_2) = \frac{\partial f(x_1,x_2)}{\partial x_1}$ represents the *local effect* of x_1 on f at (x_1, x_2), and $x_{\min,1}$ is some value chosen near the lower bound of the support of p_1 (e.g. just below the smallest observation of a feature 1)

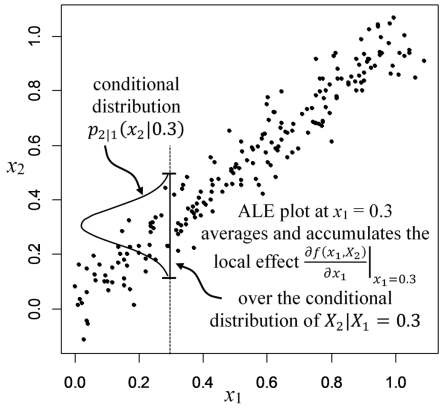
Authors state that "the function $f_{1,\mathrm{ALE}}(x_1)$ can be interpreted as the *accumulated local effects* of x_1 in the following sense. We calculate the local effect $f^1(x_1, x_2)$ of x_1 at $(x_1 = z_1, x_2)$, then average this local effect across all values of x_2 with weight $p_{2|1}(x_2|z_1)$ and then accumulate/integrate this averaged local effect across all values of z_2 up to x_1. Using conditional distribution avoid the extrapolation, similar to M plots, however, by averaging (across x_2) and accumulating (up to x_1) the local effects, ALE plots avoid the omitted nuisance variable bias that renders M plots of little use for assessing the main effects of the predictors." Of course, the idea can be extended to more than two features. The illustration of the function computation is shown in Figure 2.3a.

Estimation

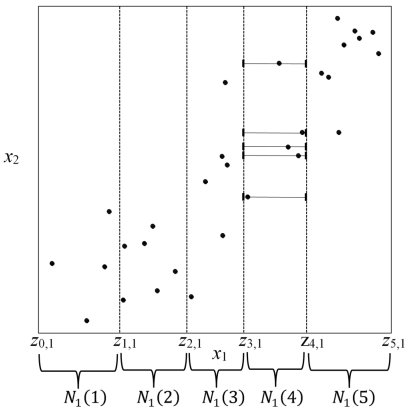
To estimate the function, we first need to estimate the uncentered effect $g_{j,ALE}$

$$\hat{g}_{j,ALE}(x) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{\{i:x_{i,j} \in N_j(k)\}} [f(z_{k,j}, \mathbf{x}_{i,j}) - f(z_{k-1,j}, \mathbf{x}_{i,\setminus j})]$$

for each $x \in (z_{0,j}, z_{K,j}]$. Here, $\{N_j(k) = (z_{k-1,j}, z_{k,j}] : k = 1, 2, \ldots, K\}$ is a sufficiently fine partition of the sample range $\{x_{i,j} : i = 1, 2, \ldots, n\}$ into K intervals. Authors have chosen



(a) Illustration of the computation of $f_{1,ALE(x_1)}$ at $x_1 = 0.3$



(b) Illustration of the notation and concepts in computing the ALE main effect estimator $\hat{f}_{j,ALE}(x_j)$ for $j = 1$ with $d = 2$ predictors.

Figure 2.3 Illustrations for the computation and approximation of ALE function. Taken from [11].

$z_{0,j}$ as a value just below the smallest observation, and $z_{K,j}$ to be the largest observation; $z_{k,j}$ was chosen as the $\frac{k}{K}$ quantile of the empirical distribution $\{x_{i,j} : i = 1, 2, \ldots, n\}$; $n_j(k)$ is then number of instances in k-th partition. Finally, $k_j(x)$ denotes the index of the interval which x falls into.

The main effect is then obtained by subtracting an estimate of $\mathbb{E}[g_{j,ALE}(X_j)]$:

$$\hat{f}_{j,ALE}(x) = \hat{g}_{j,ALE}(x) - \frac{1}{n}\sum_{i=1}^{n} \hat{g}_{j,ALE}(x_{i,j})$$

Figure 2.3b illustrates the notations used for estimation of ALE function.

2.6.4 Permutation Feature Importance

Oftentimes, instead of looking at the relationship plots, we want to know what features affected the model the most. This measure is called Feature Importance and it is often displayed as a bar plot (see the example in Figure 2.4).

One of the ways how to calculate it is by using Permutation Feature Importance (PFI) [12]. It is a method that calculates global feature importance for each feature as a decrease in a model score when a single feature value is randomly shuffled. The higher the loss in performance after perturbing feature values, the higher its importance. Usually to measure the model score we would use some loss function, depending on our problem (e.g. R2 or RMSE for regression) [2].

Here, random shuffling of feature values is done with the purpose of somewhat removing this feature and its possible interactions with other features to see how it affects the model performance. If after the shuffle the model's prediction doesn't change, we can say that the model doesn't really rely on this feature. On the contrary, if the prediction changes, the model relies on it. The bigger the change, the more the reliance.

Permutation Feature Importance is a good and intuitive measure if you want to see which features affect the model (specifically, its performance) the most. However, it might not always be very interpretable. Given the feature importance value only by itself, we can't get much from it: it tells us that the mean error in the loss function after permuting feature values is some feature importance value. As we will see in the later sections, there are methods that give us not

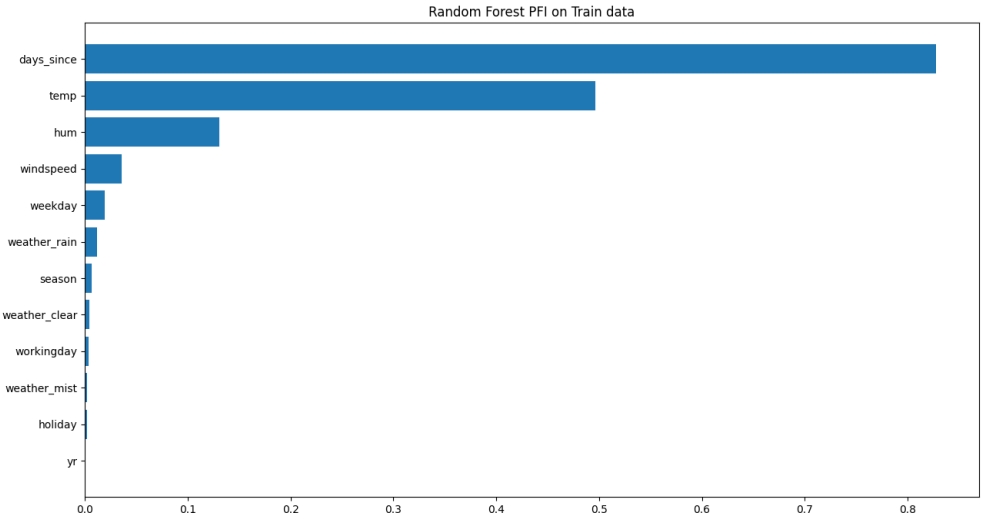


Figure 2.4 Example: permutation feature importance bar plot for a Random Forest model trained on the Bike Sharing dataset[4]. The most important feature to predict the number of bicycles rented in a given day is number of days since the starting date and day's temperature

only feature importance but also feature attribution: how much did the feature value for this specific instance changed the prediction (in the scope of the global methods, it would be mean feature attribution). Moreover, for PFI to compute the change in performance score, we need to have access to the real target values, which is not always available.

`Scikit-learn` [10] package supports this method[5].

2.6.5 Global Surrogate

"A global surrogate model is an interpretable model that is trained to approximate the predictions of a black-box model [2]". Global surrogate is not an explainability method, but an approach any ML engineer can follow. It consists of (1) selecting an interpretable model, such as linear regression or decision tree, and (2) deciding whether it approximates the black-box model well. This decision always depends on a task at hand. From the first glance, global surrogate model is not very useful model since it might be difficult to approximate complex black-box model with the interpretable one. But it's worth to try and see, if the performance difference is not huge, one might actually replace the black-box with the surrogate. If, however, the surrogate model does not approximate the black-box well, there is a trick, which lies in narrowing the scope of approximation onto one prediction only and trying to explain it using an interpretable model. This is exactly what Local Interpretable model-agnostic explanation (LIME [7]) method does. We will discuss this method in the next section shortly, but for now, let us consider that any machine learning model can be viewed as a function of multiple variables: such model (function) takes an input of several features (variables) and outputs a prediction. Using global surrogate model to explain a black-box is similar to approximate the complex function with an easier one. Most likely, it will not succeed. However, we know that if for a given data point we take neighbourhood small enough, we can approximate the complex function in this neighbourhood quite well using only linear function. This is a technique used in LIME, which we will be examining in the next section.

[4] `https://github.com/christophM/interpretable-ml-book/blob/master/data/bike.csv`
[5] `https://scikit-learn.org/stable/modules/permutation_importance.html#id2`

2.7 Local Model-Agnostic Methods

Since LIME method is not global, but local, let us first provide a brief outline of Local Model-Agnostic methods. Local Model-Agnostic methods are methods that explain single prediction for any given model. The reason for it is that they don't need an access to the model's inner parts, they treat any model as a black-box and analyze only how different inputs affect its outputs [2].

In the following sections, we will be describing and comparing two widely used local methods: SHAP and LIME. They both produce local feature importances which tell us how relevant each feature is for the model's prediction for a given instance. Of course, they do so in a different way: while LIME builds an interpretable model to imitate a black-box in the neighbourhood of a given instance which is then used to explain the prediction, SHAP shows how much of the prediction value was contributed by each feature by approximating Shapley values from the Game Theory.

Without a doubt, there are more local methods out there and some of them produce different types of output. Counterfactual Explanation is an explanation method providing us with (minimal) changes to the input to achieve the desired output [2]. For example, consider loan prediction problem: given individual's data what is the probability that he will get the banking loan? Naturally, understanding the reasons behind a person's approval or denial of a loan is essential and we can find it out by using feature importance techniques. But if, say, a person did not get a loan, it would also be valuable to know what this person could change to get it, which is a task of counterfactual explanations which we will not explore in this work, but we found it necessary to mention.

2.7.1 LIME

Local Interpretable Model-Agnostic Explanations (LIME) builds an interpretable surrogate model that tries to approximate a black-box model but only in the vicinity of the given data instance. It was introduced in the "“Why Should I Trust You?” Explaining the Predictions of Any Classifier" paper from 2016 where authors come up with this method to explain individual predictions and compare it with existing ones from that time showing that LIME surpasses them [7, 2].

Practically, LIME is a framework which searches for an interpretable model g (here, we also call this model an explanation) which optimizes an equation which includes two key features of an explanation outlined by the authors: *interpretability* and *local fidelity*. Here, interpretability is a measure of how complex or explainable the explanation is: the more complex it is, the less explainable it is. For linear models it could be a number of non-zero weights. Local fidelity is a measure telling how well the explanatory model approximates given machine learning model in the vicinity of the predicted instance. The equation we want to optimize is as follows:

$$\xi(x) = \underset{g}{\operatorname{argmin}}\, L(f, g, \pi_x) + \Omega(g)$$

where $\xi(x)$ is the produced explanation, g is interpretable model, π_x is a function measuring proximity within x, $\Omega(g)$ measures complexity of a model g, and, finally, $L(f, g, \pi_x)$ is fidelity function which measures how well model g approximates model f.

Surely, authors provide concrete implementation of this framework which uses sparse linear model with L1 regularization (Lasso) as an interpretable model type, exponential kernel as a proximity measure and *locally weighted square loss* as fidelity function [7]. Exponential kernel is defined as:

$$\pi_x(z) = e^{\frac{-D(x,z)^2}{\sigma^2}}$$

where D is the distance function, σ is kernel width hyperparameter specified by the user. With this parameter we can change the neighbourhood of the instance. Locally weighted square

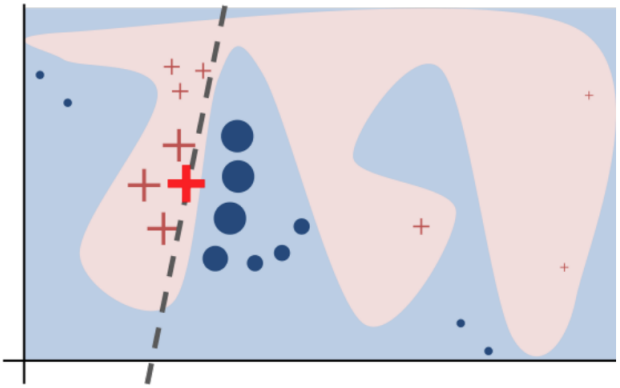


Figure 2.5 "Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful." Taken from [7].

loss fidelity function is then defined as:

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

where z is the perturbed instance based on the input x from the original space $\mathbb{R}^d$ used by the model f, z' is the *interpretable representation* of z in the binary space $\{0,1\}^{d'}$, $\mathbb{Z}$ is the new perturbed dataset, and g is the linear model with weight vector w_g and $g(z')$ is defined as $w_g \cdot z'$.

Essentially, the algorithm first creates new data in the proximity of the given instance by perturbing it. Then it weights points in generated data depending how close it is using the proximity function: the closer, the more weight it has. Finally, it trains a Lasso model on this new data to minimize the weighted square loss with the desired number of features K. For illustration, see Figure 2.5. By selecting fewer features, the explanation becomes more interpretable, but less accurate: the result is a shorter explanation with a small number of features which is less accurate. This, as we will see in the next section, is quite opposite to the explanation given with SHAP method. Authors also provide an implementation of this algorithm for python in the `lime` package [6]. It can be used for tabular, text and image data.

LIME creates explanations in rather intuitive way, but its biggest shortcoming is that selecting different kernel width parameter σ results in different explanations which is not desirable. We can't visualize or measure how well the produced model approximates our black-box in the vicinity of the instance we want to explain (in case we have more than 2-3 features, which is almost always the case). In the illustrative example from Molnar (Figure 2.6) we can see that given a non-linear function of 1 variable (feature) and point x we can get negative, zero and positive feature effects depending on the kernel width value. Moreover, it is arguable, that applying the same vicinity function for all features makes sense: some feature could have bigger or smaller impact on measuring the vicinity [2].

In the practical part, we will try to see the examples on real data when and how LIME fails or succeeds to explain single predictions and compare them to the mentioned SHAP method.

[6] `https://github.com/marcotcr/lime`

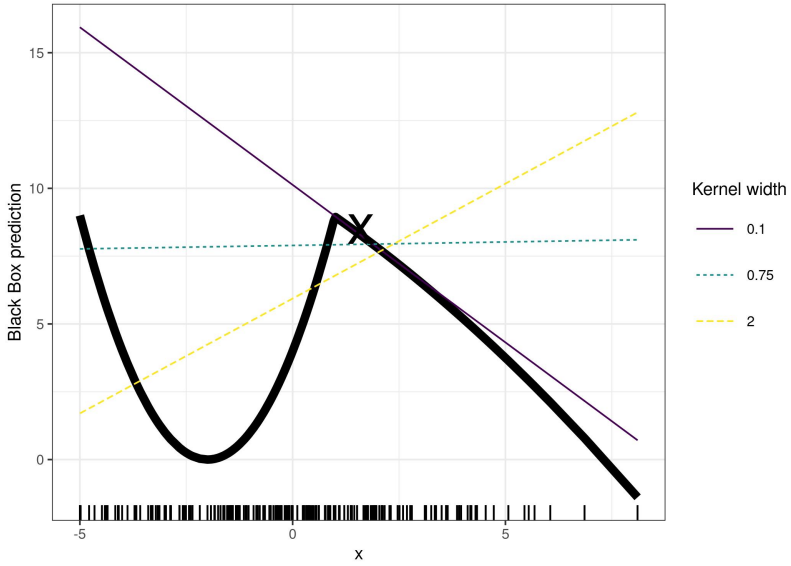


Figure 2.6 Explanation of the prediction in $x = 1.6$ of a black-box function (black) using LIME and different kernel width σ values. Feature importance is given as a slope of a linear function: for $\sigma = 0.1$ importance is negative, 0.75 - zero, 2 - positive. Taken from [2]. (note: computed using existing python implementation of LIME provided by the LIME authors)

2.7.2 Shapley Values

Shapley values come from the Game Theory as a way to fairly distribute the payoff in the cooperative game with p players [13]. For example, imagine a company with p employees. This company has a goal to fairly distribute its profits, \$1,000,000, between its p employees. For the illustrative purposes, the company knows how much it would make for any group of employees (any possible subset of employees). To make a fair pay out to the employee e_i the company would calculate the difference between how much it would make with and without the employee e_i over all the possible employee groups and take a mean value of it [14].

More formally, we have a cooperative game w with p players and a characteristic function v: $2^P \rightarrow R$ mapping all the subsets of players to the real number. For the coalition of players S characteristic function v describes the total expected sum of payoffs the players of S can obtain by cooperation.

The Shapley value formula for player i is:

$$\phi_i(v) = \sum_{S \subseteq P \setminus \{i\}} \frac{|S|!(p - |S| - 1)!}{p!}(v(S \cup \{i\}) - v(S))$$

where P is the total set of players and the sum extends over all subsets S of P not containing player i, including the empty set.

The Shapley Value ϕ_i is a "fair" way to distribute total gains to the player i in the sense that is it the only distribution satisfying the properties, or axioms, listed below [15].

Efficiency The sum of the Shapley values of all players equals the value of the grand coalition, so that all the gain is distributed among the players:

$$\sum_{i \in P} \phi_i(v) = v(P)$$

Symmetry The contributions of two players i and j should be the same if they contribute equally to all possible coalitions. If

$$v(S \cup \{i\}) = v(S \cup \{j\})$$

for all

$$S \subseteq \{1, \ldots, p\} \setminus \{i, j\}$$

then

$$\phi_i = \phi_j$$

Linearity If two coalition games described by gain functions v and w are combined, then the distributed gains should correspond to the gains derived from v and the gains derived from w:

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w)$$

$$\phi_i(av) = a\phi_i(v)$$

for every $i \in P$ and any real number a.

Dummy (or Null Player) A player i that does not change the predicted value – regardless of which coalition of players it is added to – should have a Shapley value of 0. If

$$v(S \cup \{i\}) = v(S)$$

for all

$$S \subseteq \{1, \ldots, p\}$$

then

$$\phi_i = 0$$

Shapley values in xAI

Now that we now what Shapley values are, we can realize that we can view ML model, or its prediction function f, as a payoff function, and a single instance with p feature values as p players in the coalition game. In that case, Shapley value $\phi_i(f)$ for instance x and feature value x_i can be considered a feature importance - or better even, an attribution, as Shapley values satisfy the *efficiency* axiom: the sum of attributions should be equal to the prediction value. The only thing missing from the computation of Shapley values is the way to compute the prediction of the instance with missing features. To estimate it, we can simply replace the values of missing features with the values of a randomly sampled instance from the dataset.

The main problem of the Shapley values method is that it is very computationally inefficient: we need to go through every subset of P features and calculate the difference in prediction function with and without given feature i. This scales exponentially resulting in $O(2^P)$ complexity for an attribution of a single feature.

The more effective way to calculate it is by Monte-Carlo sampling introduced in [16]. The overall complexity of it is $O(M)$ where M is chosen in advance. It is important to note that there is no optimal way to choose M. Both approximations, however, suffer from creating and evaluating on the unrealistic datapoints in case of correlation: if the feature is correlated with others, replacing their values with the ones from randomly sampled instances (that is features' marginal distributions) may lead to the data points that are (1) improbable and (2) on which the model was not trained.

2.7.3 SHAP

SHAP (SHapley Additive exPlanations) [6] is a method that efficiently approximates Shapley values. Moreover, it connects LIME and Shapley values by representing an explanation as an additive feature attribution method, a linear model like so:

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

where g is the explanation model, M is number of features, z is the coalition vector $\in \{0,1\}^M$, ϕ_j is feature attribution, Shapley value, for feature j.

In other words, the explanation result of the SHAP is not only approximate Shapley values, but a linear model where feature weights are the appropriate Shapley values. Note that for the "empty" coalition vector $\{0,0,\ldots,0\}$, $g(z) = \phi_0$. In the Game theory this ϕ_0 is the fair pay off when no player is playing. For SHAP, ϕ_0 is the intercept of the resultant linear model.

SHAP satisfies all the properties of Shapley values and also a number of others specified in the original paper that are used to simplify calculations.

2.7.3.1 KernelSHAP

KernelSHAP is a model-agnostic SHAP method. As described in [2], the algorithm of approximating Shapley values with KernelSHAP is following:

Given an instance x with M features, number K and prediction function f:

1. Sample K coalitions $z_k \in \{0,1\}^M$ (where 1 is "feature present", 0 is "feature missing")
2. For each coalition sample z_k find the prediction by first converting coalition to the original feature space $f(h_x(z_k))$
3. For each coalition sample compute the weight with the SHAP kernel
4. Fit weighted linear model
5. Return Shapley Values ϕ_k, feature weights of the fitted model

h_x is a function that maps a binary coalition vector to the original feature space of x and is defined as follows:

$$h_x(z_k)_i = \begin{cases} x_i, & \text{when } z_{ki} = 1 \\ r_i, & \text{when } z_{ki} = 0, \text{where } r \text{ is randomly sampled instance} \end{cases}$$

It follows the same problem as all the other perturbation methods: it breaks the feature dependences in case of correlation.

Even though the output of both SHAP and LIME is a linear model, the kernel function is what differentiates these methods so much. Kernel function for KernelSHAP is SHAP kernel which is defined as:

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

The idea behind it is that coalitions with 1 or 2 features give much more information than, for example, coalitions with only half the features present, and, therefore, get substantially more weight. In fact, as we see in Figure 2.7, SHAP kernel gives the minimum weight when coalition includes only half of the features. Similarly, for cases when only 1 or 2 features are missing: they give more information.

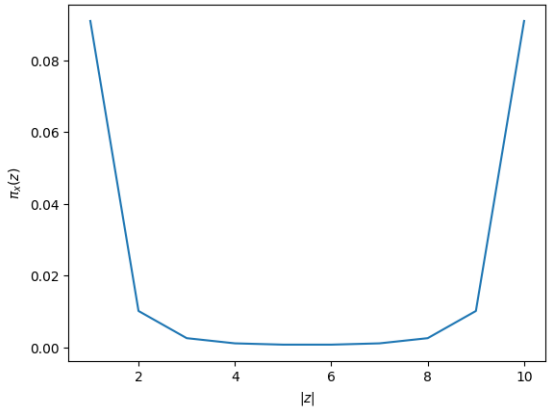


Figure 2.7 SHAP kernel function for $M = 11$

With this in mind, more practical way to sample coalitions is to first sample M coalitions that miss one feature, and M coalitions missing all but one, then $M(M-1)$ coalitions missing 2 features and $M(M-1)$ coalitions missing all but 2, until the desired K is achieved.

Another benefit of KernelSHAP comes from the fact that the Shapley values are the feature weights of the linear model trained to minimize the quadratic loss function. We can utilize not only simple linear regression model, but also add different regularization terms such as L1, which would result in a sparse model and, hence, sparse explanation. This technique would, however, yield Shapley values different from the real ones.

2.7.3.2 TreeSHAP

TreeSHAP is SHAP method specific to tree and tree ensemble models, such as Random Forest, XGBoost [17], CatBoost[7]. The main difference with KernelSHAP is that TreeSHAP uses not the marginal, but conditional distribution of a feature which actually changes the game features play. This violates the null player axiom because the feature effect is being distributed among correlated features and features that are not used by the model can get non-zero Shapley values.

However, TreeSHAP is significantly faster in computing approximate Shapley values than KernelSHAP and, especially, exact Shapley values. The intuition of how TreeSHAP computes the expected prediction for a single tree, instance x and feature coalition S is as follows:

1. If S includes all the features, then the expected prediction is the prediction in the node that x fell into.
2. If S includes no features, then expected prediction is the average over all the terminal nodes of a tree.
3. Otherwise, expected prediction is the (weighted) prediction over all reachable nodes, where unreachable nodes are those which decision path contradicts values in x_S.

Thanks to the linearity axiom of Shapley values, for ensemble methods we can add the values together to get the resultant Shapley value. [2]

We have to note, that, even though TreeSHAP utilizes conditional distribution, its implementation in `shap` package also support marginal version by specifying `feature_perturbation` parameter: *"Since SHAP values rely on conditional expectations, we need to decide how to handle correlated (or otherwise dependent) input features. The "interventional" approach breaks the dependencies between features according to the rules dictated by causal inference (Janzing et al.*

[7] `https://catboost.ai/en/docs/`

2019). Note that the "interventional" option requires a background dataset data, and its runtime scales linearly with the size of the background dataset you use. Anywhere from 100 to 1000 random background samples are good sizes to use. The "tree_path_dependent" approach is to just follow the trees and use the number of training examples that went down each leaf to represent the background distribution. This approach does not require a background dataset, and so is used by default when no background dataset is provided." [8]. This makes TreeSHAP very versatile, since not only it speeds up the computation of Shapley values comparing to KernelSHAP, but also we as users can choose how to deal with feature dependences.

2.7.3.3 Other SHAP explanations

Major advantage of the SHAP methods is their versatility. SHAP methods, apart from providing feature attributions for local explanations, also provide a number of global explanations that are based on the local ones across the examined data. The result is a vast framework[9], which includes a number of explanation results, both local and global, that is unified under one theoretical domain - Shapley values. The small downside is that one must examine the framework beforehand, at least get the basics, before concluding results from the explanations generated.

Here are a few of the explanations available in the package right now:[10]

Feature Importances Global SHAP feature importances are just the average absolute values of SHAP values over all SHAP values (which are computed for either all or a sample of data). They can be then displayed as a regular bar plot.

Summary (beeswarm) Plot The beeswarm plot shows a summary of how the top features in a dataset impact the model's output. It resembles Feature Importance bar plot, but for each feature beeswarm also shows how low and high values of this feature affect the SHAP values.

Dependence Plot Dependence Plot is a plot similar to PDP: it plots the relationship plot between a given feature and target value based on the SHAP values. It does so by simply plotting all $(x_{i,j}, \text{SHAP value}_{i,j})$ points in $\{(x_{i,j}, \text{SHAP value}_{i,j}) : i = 1, 2, \ldots, N\}$ for feature j.

2.8 The Disagreement Problem

2.8.1 Introduction

"The disagreement problem" is a problem when different explanation methods produce different explanations. Consider yourself having a complex black-box model that gives some output that we, for some reason, say, the output turns out to be quite controversial to what we might have expected, want to explain. For this task, popular SHAP and LIME methods could be used. It can happen that SHAP and LIME will output different explanations, transforming the task into finding which explanations to trust more. Now the question is how to do it. This is the "disagreement problem".

The problem itself comes from the paper of the same name [4] in which authors define the problem, survey a number of practitioneers on the topic and based on it provide ways to evaluate the degree of disagreement between explanations. Important to mention is that authors do not solve the problem, i.e. how, given 2 explanations from 2 different methods, you can choose the "correct" one. Instead they (1) formally define the problem (in the realm of local post-hoc

[8] `https://shap.readthedocs.io/en/latest/generated/shap.TreeExplainer.html`
[9] `https://shap.readthedocs.io/en/latest/`
[10] The full list with examples and usages is available in the SHAP documentation: `https://shap.readthedocs.io/en/latest/api.html#plots`

explanation methods), (2) survey practitioneers (details described later), and (3) show why "the Disagreement Problem" actually is a problem. Namely, how often it occurs with some of the most used explainability methods.

Two insightful parts of this work were (1) vast empirical analysis and (2) survey of ML engineers on how often they deal with disagreements in xAI and how the deal with it if they do.

We are thinking this paper is quite insightful, describing the current state of the problem, which so far was not in the limelight of xAI, but is definitely going to be in the coming years, which is why we would like to share as many insights from it as possible in this section.

First, we will take a look into the survey the authors made.

2.8.2 The origin of the problem: "Disagreement Problem" paper

Survey and its results

The survey included 25 engineers (data scientists and ML engineers), all of whom have already been working with explainability for extended periods of time. Each of them was asked a series of questions, which included the following: Q1) *How often do you use multiple explanation methods to understand the same model prediction?* Q2) *What constitutes disagreement between two explanations that explain the same model prediction?* Q3) *How often do you encounter disagreements between explanations output by different methods for the same model prediction?*

The results are presented in the following subsection.

Most important metrics from the survey

Top features are different: "21 of 25 participants mentioned that a set of top features (features which importance value contributed the most) is 'the most critical piece of information' that they rely on in their day-to-day workflow. When two explanations have different sets of top features, they consider it to be a disagreement."

Ordering among top features is different: 18 out of 25 participants (72%) in our study indicated that they also consider the ordering among the top features very carefully in their workflow. Therefore, they consider a mismatch in the ordering of the top features provided by two different explanations to be a disagreement.

Direction of top feature contributions is different: 19 out of 25 participants (76%) mentioned that the sign or direction of the feature contribution (is the feature contributing positively or negatively to the predicted class?) is another critical piece of information. Any mismatch in the signs of the top features between two explanations is a sign of disagreement. As remarked by one of the participants, "I saw an explanation indicating that a top feature bankruptcy contributes positively to a particular loan denial, and another explanation saying that it contributes negatively. That is a clear disagreement. The model prediction can be trusted with the former explanation, but not with the latter.".

Relative ordering of certain features is different: 16 of our study participants (64%) indicated that they also look at relative ordering between certain features of interest; and if explanations provide contradicting information about this aspect, then it is considered a disagreement. For example, one of the participants remarked, "I often check if salary is more important than credit score in loan approvals. If one explanation says salary is more important than credit score, and another says credit score is more important than salary; then it is a disagreement."

2.8.2.1 Disagreement Metrics

Metrics formalized

Based on the survey responses authors formally define 6 metrics to measure the (dis)agreement between local explanations:

Feature Agreement: computes the fraction of common features between the sets of top-k features of two explanations. Given 2 explanations E_a and E_b, number of features k, it is formally defined as:

$$\text{FeatureAgreement } (E_a, E_b, k) = \frac{|\ \text{top-features } (E_a, k) \cap \text{ top-features } (E_b, k)\ |}{k}$$

Rank Agreement: computes the fraction of features that are not only common between the top-k features of two explanations, but also have the same position in the respective rank orders. Given two explanations E_a and E_b, it can be formally defined as:

$$\frac{|\bigcup_{s \in S} \{s \mid s \in \text{ top-features } (E_a, k) \wedge s \in \text{ top-features } (E_b, k) \wedge \text{rank}\,(E_a, s) = \text{rank}\,(E_b, s)\}\ |}{k}$$

Sign Agreement: computes the fraction of features that are not only common between the top-k features of two explanations, but also have the same sign in both explanations. To define formally:

$$\frac{|\bigcup_{s \in S} \{s \mid s \in \text{top-features}\,(E_a, k) \wedge s \in \text{top-features}\,(E_b, k) \wedge \text{sign}\,(E_a, s) = \text{sign}\,(E_b, s)\}\ |}{k}$$

Signed Rank Agreement: This metric fuses all the above notions, and computes the fraction of features that are common between the top-k features of two explanation and have the same sign and position in both explanations. Formally:

$$\frac{\begin{array}{l}|\ \bigcup\limits_{s \in S} \{s \mid s \in \text{top-features}\,(E_a, k) \wedge s \in \text{top-features}\,(E_b, k) \\ \wedge \text{sign}\,(E_a, s) = \text{sign}\,(E_b, s) \wedge \text{rank}\,(E_a, s) = \text{rank}\,(E_b, s)\}\ |\end{array}}{k}$$

Rank Correlation: measures agreement between feature rankings for a selected set of features F using Spearman's rank correlation coefficient. Formally:

$$RankCorrelation\,(E_a, E_b, F) = r_s\,(Ranking\,(E_a, F)\,, Ranking\,(E_b, F))$$

Pairwise Rank Agreement: Pairwise rank agreement takes as input a set of features $F = \{f_1, f_2, \dots\}$, that are of interest to the user, and captures if the relative ordering of every pair of features in that set is the same for both explanations:

$$PairwiseRankAgreement\,(E_a, E_b, F) = \frac{\sum_{i,j \text{ for } i<j} \mathbb{1}\,[\text{ RelativeRanking }\,(E_a, f_i, f_j) = \text{RelativeRanking}\,(E_b, f_i, f_j)]}{\binom{|F|}{2}}$$

where RelativeRanking(E, f_i, f_j) is an indicator function which returns 1 if feature f_i is more important than feature f_j according to explanation E, and 0 otherwise.

2.8.2.2 Empirical analysis

Having defined metrics to measure the disagreement between 2 local explanations, authors then decide to compare 6 widely used explainability methods on different ML models and different data to see how often on average they disagree.

In their analysis, specifically on tabular data, authors are comparing 6 different explanation methods on 4 ML models that were trained on 2 different datasets. Datasets included COMPAS [18] and German Credit dataset [19]. Four ML models analyzed were: logistic regression, densely-connected FFNN, random forest, gradient boosted trees. 6 explanation methods were: 2 perturbation-based (KernelSHAP and LIME) and 4 gradient-based (Vanilla Gradient, Gradient*Input, Integrated Gradients and SmoothGrad). In this work, we will be primarily focusing on the results for SHAP and LIME methods because they are the ones we are most familiar with.

The average disagreement for the FFNN model trained on COMPAS dataset between 6 explainability methods from above is summarized in Figure 2.8. Looking at SHAP and LIME, we can see that:

- average feature agreement is very high, taking in mind that it's high for every pair of methods, which doesn't say much,
- average rank agreement shows value of 0.381, the value higher than for pairs of methods showing no agreement but less smaller than for pairs of methods showing higher (almost 1) agreement,
- average sign agreement shows significantly higher results, but considering almost 1 value for feature agreement (5 top features are almost always the same in the explanations) that shows that close to half of the feature importances for the same feature in SHAP and LIME explanations have different signs, which we would argue not desirable,
- average signed rank agreement: 0.226, smaller value than regular rank agreement that again shows that signs of feature importances are often different,
- metrics that treat explanation as a sequence – rank correlation and pairwise rank agreement – show higher agreement: 0.8 correlation value is high enough to suppose that correlation exists; but we have to keep in mind that (1) for these metrics all 7 features are included and (2) 7 numbers of a sequence might not be enough to draw definitive conclusions.

Additionally, in Figure 2.9 we can observe how increasing number of selected features k affects average disagreement between SHAP and LIME methods:

- average rank agreement value of 0.485 for 1 feature tells us that in half of the cases the most important feature in 2 explanations is not the same,
- this value drops a bit for k=4 and increases by little for k=7,
- the same tendency we see for average signed rank agreement but with much less agreement: 0.285 for 1 feature and 0.187 for optimal number of features, 4.

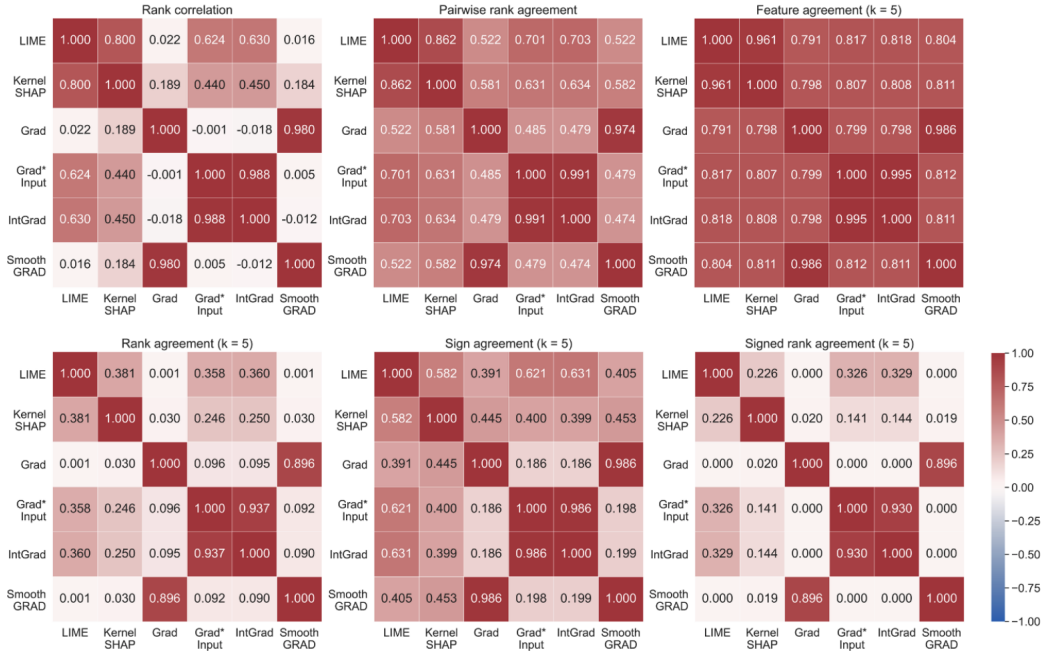


Figure 2.8 "Disagreement between explanation methods for neural network model trained on COMPAS dataset measured by six metrics: rank correlation and pairwise rank agreement across all features, and feature, rank, sign, and signed rank agreement across top $k = 5$ features. Heatmaps show the average metric value over test set data points for each pair of explanation methods." Taken from [4].

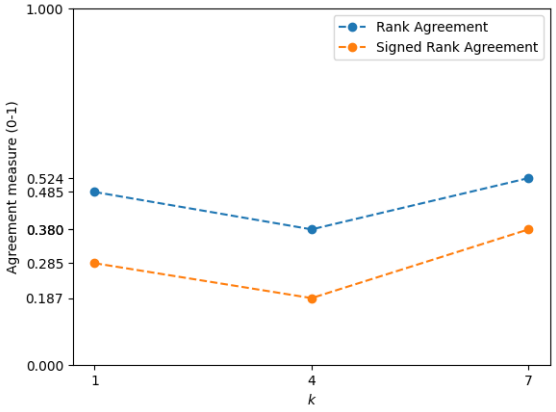


Figure 2.9 Effect of number of features included in explanation on "Rank Agreement" and "Signed Rank Agreement" disagreement metrics [4].

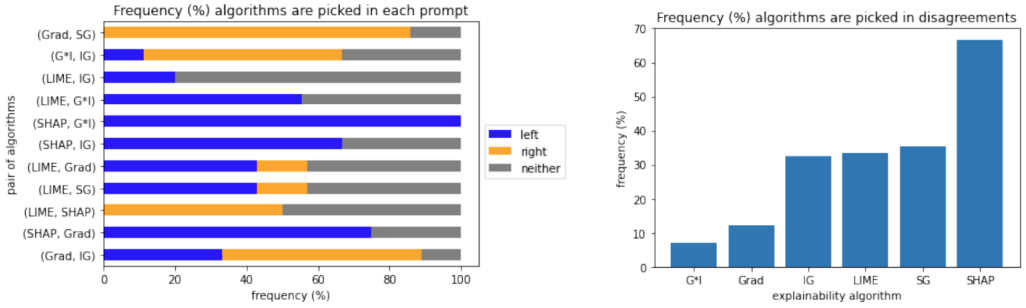


Figure 2.10 Frequency of explainability methods picked. When given 2 explanations produced by LIME and SHAP (left), practitioners were prone to pick SHAP more often. SHAP was chosen most often overall (right) [4].

Obviously, one trained model on one dataset only is not enough to draw conclusions, however we argue that for this specific case results given by SHAP and LIME are quite controversial. And that with the COMPAS dataset being very small for the real world: one would wonder what happens on more complex data.

Taking into account survey from the beginning (summarized in subsection 2.8.2), and analysis on real data, we are now coming the last part of the paper, where authors conduct a user study that shows examples of how practitioneers solve disagreements on the data from empirical analysis part.

2.8.2.3 User study and its insights

First, let us describe how the study was conducted and condense its insights next.

The study included 25 participants answering a set of questions based on the setup from the empirical analysis: neural network trained on the tabular COMPAS data. Each participant was briefed with data and model used. Then, each participant was shown a series of 5 prompts, each prompt presented two explanations of FFNN model's prediction corresponding to particular data instance from the COMPAS dataset. For each prompt participant was asked to answer the following question: "To what extent do you think the two explanations shown above agree or disagree with each other?"; and if the answer indicated any level of disagreement, second question followed: "Since you believe that the above explanations disagree (to some extent), which explanation would you rely on?". For each questions, participants also had an option to give a detailed written response with their reasoning.

Most of the results in this section are based on 6 different explainability methods that include KernelSHAP and LIME, unless said otherwise.

Observed disagreement frequency Significant insight was that participants observed disagreement quite often: 4%, 28%, 50% and 18% of the responses indicate *completely agree*, *mostly agree*, *mostly disagree*, and *completely disagree*. In total, 68% of the responses showed disagreement.

Were certain explanations favored over others? It turns out that SHAP was picked almost twice as often as any other method (LIME included). This we can observe in Figure 2.10 on the right. On the left we can see that while choosing between SHAP and not LIME method, participants were inclined to choose SHAP or neither; similar picture is for LIME and not SHAP method, but with lesser frequence than SHAP; and finally between SHAP and LIME, SHAP was chosen half of the times, while the half participant chose neither.

How are disagreements resolved? Based on the participants' responses, authors find 3 main themes, that dictate why participants were choosing one algorithm over the other:

1. *One method is inherently better than the other because of its associated theory or publication time (33%).*
2. *One of the generated explanations matches intuition better (32%)*
3. *LIME and SHAP are better because COMPAS dataset comprises of tabular data (23%)*

We can see that there is no systematic approach to solving disagreements. More importantly, in the themes above we can see that there is no substantial reasoning to select one method over the other. Appealing to publication time is same as appealing to novelty, which for methods such as SHAP and LIME, having a publication difference in about a year, should not necessarily even apply. Again, comparing SHAP and LIME and saying that "associated theory" for one method - say for SHAP - is better than the other might also not be a case, because they are inherently different, and comparing them is a difficult task. Having metrics, such as ones defined in this paper, we can - albeit quite limitedly - do the empirical comparison of these methods - the task we will give a try in the practical part. "Better intuition match" should not necessarily hold. For example, having an overfit (or underfit) model and 2 different explanations, while deciding to choose one of them, you would be looking at how well the explanation matches your intuition, not the model you want to explain: overfit (or underfit) model definitely shouldn't match your intuition, and on the contrary, it often doesn't. "SHAP and LIME being better on tabular data" is potentially verifiable statement, however we won't be focusing on it, since we are interested in (1) differences between those methods and (2) non-tabular data is not the focal point of this work.

Disagreements in day to day work Finally, out of 25 engineers authors left 19 who have used explainability methods before in their work to understand how they treat disagreements in natural conditions: day to day work. They were asked two questions: *(Q1): Do you observe disagreements between explanations output by state of the art methods in your day to day workflow?* and *(Q2): How do you resolve such disagreements in your day to day workflow?*

14 out of 19 respondents answered *yes* to the first question. 7 out of 14 turned out to have personal heuristics to select the explainability methods and 5 said they "had no clear answer" and many of the responses showed desire for the research community to make progress in this area with phrases like "I hope research community can provide some guidance".

Once again, we can see that there is no systematic approach ML practitioneers follow to resolve disagreements: best case, they are following personal preferences, based on popularity, novelty, theory underneath the algorithms, some, however, show lack of certainty and knowledge in the area, which really does exhibit the need of further research in the field.

2.9 Impact of correlation on explainability

2.9.1 Background

Correlation in data is not an exception but a rule [3]. And correlation already creates many different problems in machine learning that we need to take into account. This leads to the question, whether correlation impacts the area of explainable AI and if so, how?

Correlation in that area is not a new problem. Many people have already pondered on this topic, and Molnar even has a sort of a "cookbook" on how to solve it that we will describe in the final section of this topic, but before let us tackle the problems that correlation can lead to.

On general, the problem of correlated data in the xAI can be boiled down to the question how different explainability methods can "deal with it", namely if a method has some theory

underneath that doesn't work well with correlation. There is a number of methods that in order to create explanations permute feature values of data points, which, provided features are dependent, leads to the new improbable data points that are used by the method to generate an explanation. Example could be a KernelSHAP, a model-agnostic Shapley Values approximation explainability method, that to approximate the feature attributions assumes that all the features in the data are linearly independent. Or Partial Dependence Plot: it creates a plot showing a *marginal* dependence of one given feature on the output variable. For the given value of a feature, PDP assumes that every point in the data has this value of this feature and calculates the average prediction on this new, artificial dataset. The problem occurs when the feature we are examining is correlated with the others - this way it is improbable that data points could have this feature value.

Consider, for example, bike rentals dataset with a season feature to draw a plot for and other features that include temperature. In this case, PDP will be inputting data points with season Winter and temperature 15, 20, 25° to the ML model - points (1) highly improbable, therefore not useful to take into consideration, and (2) on which the model was definitely not trained, and therefore on which prediction could be very misleading. Such data generation technique could be called *marginal* - given feature is assumed to be independent to the others, its marginal distribution is used. Instead, there are methods that generate the data points using feature's *conditional* distribution. In the latter, points with winter season will only have "real" temperature values, which is achieved by also changing correlated features according to conditional distribution. This, in turn, can also give incorrect attributions since single feature explanation would then include effects of other correlated features. Without giving this issue much thought, one could draw conclusion that, given that data is correlated, conditional methods are better than marginal ones since latter might compute their effects on data instances outside of manifold causing them to be incorrect.

This is the problem of "True to the data VS True to the model" explanation - a topic that comes down to one simple question - "What is the goal of your explanation?"

2.9.2 "The goal of explanation"

In the "True to the Model or True to the Data?" [20] article authors state that the choice of marginal or conditional "version" of SHAP method is application-dependent as opposed to "conditonal version being more preferable in general" - the conclusion of their previous work. Here, application is whether an explanation should be "True to the Model" or "True to the Data". It has been safely extended to other explainablity methods outside of SHAP [3].

But first, we will define the meaning of marginal and conditional methods. These namings are borrowed from a series of Molnar's articles [3]. Often, marginal "version" of a method also has a conditional counterpart. For instance, Partial dependence plot and M Plot - PDP being marginal, M Plot being conditional. We also need to note that not all methods could fall into the dichotomy of marginal and conditional. Some methods - such as Influential Functions - are simply not affected by the correlation.

Marginal methods: methods that utilize marginal distribution of a feature; it is assumed that it is independent from other features. The advantage of these methods is that the interpreting such explanation is very simple: it gives an effect of a single feature, not more not less. On the other hand, the problem is that these methods rely on "improbable data instances" to generate an explanation resulting in incorrect effects.

For marginal methods or explanations another name could be used - *interventional explanations* - they intervene in the feature's conditional distribution by assuming its independence.

Examples of marginal methods include PDP, Permutation Feature Importance.

Conditional methods: methods that in turn utilize conditional distribution of a feature. As opposed to marginal methods, if generate data, conditional methods will do so with respect to conditional distribution of a feature by changing correlated features as well as given. This technique prevents evaluating a model on the "improbable data instances", however, it also doesn't guarantee "real" feature effects. For example, in sparse models conditional methods could give non-zero feature effects to the features that are not used by the model at all.

Conditional explanations are also called observational explanations in the paper.

Examples of conditional methods include M Plot, Conditional Feature Importance.

True to the Data vs True to the model [20] So how does selecting marginal or conditional method depend on the goal of our explanation? Consider two real world examples: (1) credit card default prediction and (2) problem in cancer biology of which genes' expressions determine particular outcome. In the first example, the goal could be to explain why your customer did or did not receive a loan (default) and what can be done to increase the chances. In this case, we concentrate only on the model and why it gives an output it gives. To do just that, authors use Shapley values to select features that affected the default prediction the most and then change them (by mean imputing) to try to increase the odds of getting a loan. They show that by using marginal version of SHAP they were able to decrease the default chances with fewer features and higher odds difference: conditional version for a single feature was scattering the effect of this feature onto the other features, mixing and entangling effects of different correlated features in one.

Furthermore, when faced with a sparse model, marginal SHAP correctly identifies zero feature attribution for unused features which satisfies the dummy axiom. Conditional version instead mixes attributions of other correlated features into the attributions of unused features. Since, in the end, it is the model we are inspecting, this approach is *"True to the Model"*: the goal is to find effects that are, in essence, true to the model.

In the second example, the number of features (genes) could be very large in which case it's suitable to use sparse models. However, in such cases marginal methods will output zero effects for not used features which is definitely not what we want while getting insights from the real world data. Instead of using marginal methods it's more appropriate to use conditional, feature effects of which will utilize the implicit feature interactions that are hidden inside the data. Since it is the data we are investigating (albeit through the ML model), such an approach is called *"True to the Data"*. Its purpose is to gain insights of the data, not the model.

Obviously, when data is not correlated, "True to the data" and "True to the model" approaches give the same results.

2.9.3 Methods impacted by correlation

Let us now give a few examples of methods that are impacted by correlation. As we have already said in the introduction to this section, methods that generate data for the given feature to explain using the marginal distribution of this feature create or might create incorrect explanations if the data is correlated.

We have already seen that methods such as Partial Dependence Plot (PDP) and Permutation Feature Importance do exactly that.

PDP, M Plot PDP and M plot are both methods that visualize the effect of a feature. PDP uses marginal distribution, assuming that feature is not dependent on others, while M plot - conditional.

Permutation and Conditional Feature Importance Permutation Feature Importance calculates global feature importance by perturbing the feature values and measuring how this per-

turbation affects model performance as measured with some loss function. The higher the performance loss upon perturbation, the higher the importance of this feature. PFI perturbs the feature values independently for each feature. Instead, CFI perturbs feature values with respect to the conditional distribution.

KernelSHAP and TreeSHAP As we already said in Section 2.7.3 dedicated to SHAP methods, KernelSHAP assumes feature independence, while TreeSHAP does not.

2.9.4 Summary

We have shown that some pairs of methods are impacted by correlation, however this should not be an issue as choosing appropriate method in a pair could be considered application-dependent: whether the goal of explanation should be true to the model or to the data - to inspect the model or gain insights in data.

In addition, as promised in the beginning of the section, here is the recipe to deal with correlation by [3]. Yet it must be noted that this approach goes separately from "true to data/model" approach from above.

1. Check if method is impacted. If it is not, it is safe to use it.
2. Check if data is correlated. If it is not, both marginal and conditional version can be used.
3. Compare if results of conditional and marginal versions are similar. If they are, any method can be used.
4. Are you satisfied with interpreting conditional version? Conditional version makes interpretation more difficult than marginal by entangling effects of multiple correlated features.
5. Try grouping correlated features and interpreting them together in a marginal fashion. This again makes interpretation more difficult as you now create explanations for several features at once and some methods cannot show the effect for more than 2-3 features in a group (e.g. plotting methods)

2.10 Neural networks methods

In this section, we will be describing a number of model-specific methods to Neural Networks. One of them - Pixel Attribution method called Saliency Maps [9] - as the name tells, works with the image models only by highlighting the pixel areas that were used by the model for the prediction the most. Other - Influence Functions [21] - is the method that can work on both tabular and image data. It is a method of the "Influential Instances kind" that measures how important or influential particular training instance was for the model, which can be used for the more detailed analysis of the model. Last method of the same kind - Deletion Diagnostics - is in fact a model-agnostic method, but we will be showing how 2 of them are connected in the dedicated section.

In the practical part, we will be demonstrating how these methods can be used on practice with the Convolutional Neural Network model trained on the simple MNIST digits dataset. To do this we found test images that the model failed to classify and try to explain why it did - the task difficult if not impossible without these explainability methods.

2.10.1 Pixel Attributions methods

Pixel Attributions methods are methods that highlight pixels (or areas of pixels, superpixels) that were relevant for the classification of the given image. These methods include a whole

number of different methods, but, as Molnar states in his book [2], they can be primarily divided into 2 groups:

1. *perturbation-based*. These are the methods we have already met in the model-agnostic chapter - methods such as SHAP and LIME. They view a model solely as a black box or a prediction function and utilize it by perturbing the data and examining how these perturbations change the model's output.

2. *gradient-based*. These are the methods specific to the Neural Network models that utilize their internal weights by computing the gradient with respect to the input pixels. This section is dedicated to the Saliency Maps (or Vanilla Gradient) method, which is the most basic gradient-based method. Other gradient-based methods differ in the way they calculate the gradient.

2.10.1.1 Saliency Maps

Saliency Maps method introduced in the "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps" paper [9]. It "calculates the gradient of the loss function for the given class with respect to the input pixels. The output is a map of the size of the input features with negative to positive values [2]."

Basically, the whole idea is to compute a gradient of a loss function with respect to the image pixels. The algorithm to compute the gradient is backgpropagation.

The method is often called "Vanilla" Gradient as it is the most basic version of the methods computing gradient with respect to the input pixels, without any modifications. There is a number of methods that do extend Vanilla Gradient , such as DeconvNet [22] and SmoothGrad [23]; and methods that are more suitable for the Convolutional Neural Networks, such as Grad-CAM [24] that backpropagates the gradients only to the convolutional layer, often the last one. We will not be reviewing them in this work. The example of Saliency Maps from original paper is illustrated in Figure 2.11.

2.10.2 Influential Instances

Influential Instances [2] is a method that measures the influence of a particular training instance on the model. Without further ado, let us give an overview of Deletion Diagnostics method, because the whole idea of the Influential Instances can be explained on its example.

2.10.2.1 Deletion Diagnostics

For given training instance, Deletion Diagnostics computes the change in model's parameters or performance upon removing this instance from the training set and retraining the model. Influential instance is defined as an instance from the training set removing of which highly changes model's performance or parameters. Since an absolute change is not meaningful, it is a relative change that matters: we can find the influence measure for all the training instances, sort them and find most and least influential ones. The most influential ones are useful for the general model analysis and diagnostics to see what instances had the greatest impact on the model. The opposite goes for the least influential instances: these are the instances that had the least impact on it. Moreover, if the instance influence is 0 or close to it, then this instance had zero impact on the model as well as its performance. Given this insight, practitioner might decide what to do with such an instance, for example he might remove it from the training set completely, effectively reducing the data size.

Apart from providing a global explanation - the most important instances for the model - this method can also produce local explanations - the most important instances for the prediction of one particular observation - which can be highly valuable for debugging different biases as we

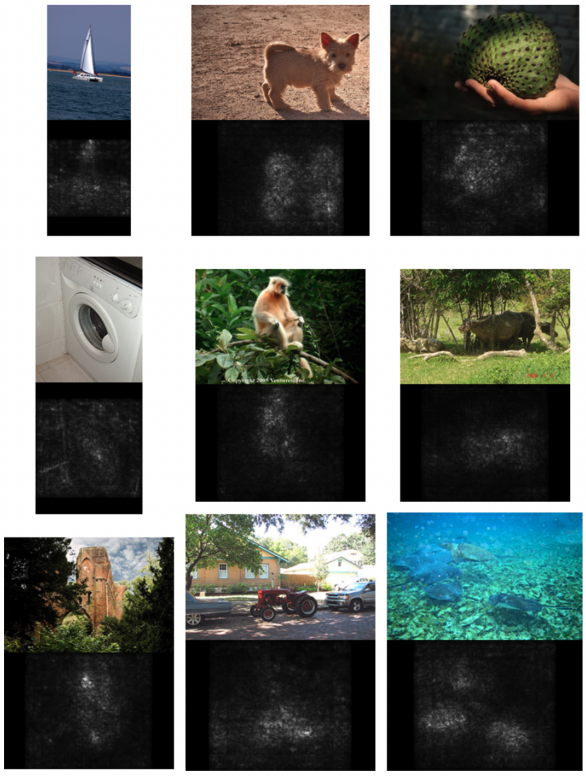


Figure 2.11 "Image-specific class saliency maps for the top-1 predicted class in ILSVRC-2013 test images. The maps were extracted using a single back-propagation pass through a classification ConvNet. No additional annotation (except for the image labels) was used in training." Taken from [9].

will see for ourselves in the practical part. As already said, the influence can be measured as a change in model's parameters - which is used for global analysis - or model's prediction. The latter is used for local explanations. Influential instances for the single prediction are instances removing of which changes the prediction of this instance the most.

It is worth noting that this method is model-agnostic and can be applied on both tabular and image data. In fact, this is not really a method, but an approach that can be easily implemented for any model by hand.

There are, however, two big differences of this method from the ones we have seen before. First, the output this method produces is always the set of instances from the training data. Explaining the model or prediction of it via a group of training observations is quite unique. This, however, is the strength of the method, since the whole goal of the machine learning is to train a model that generalizes the knowledge from the training data, and we would argue that representing this knowledge using the training instances is rather effective. Second, this approach requires retraining the model, which, controversially, does not give an explanation of the model, since the change in performance or parameters is measured between 2 different models. More appropriate would be to say that this approach measures the influence on the model *learner* - the algorithm used for training a model.

This leads us to the main problem of Deletion Diagnostics method - retraining the model for each training instance. It is evident that such approach is highly inefficient and its computational complexity grows with both the size of the training set and complexity of the model being explained [2].

The implementation of this approach which is specific to the neural network models and

which does not require the model retraining is called Influential Functions.

2.10.2.2 Influential Functions

As already said, Influential Functions [21] is a method specific to neural networks that measures instance influence on the model parameters. Instead of removing the instance as Deletion Diagnostics does it, "Influential Functions simulates its removal by upweighting the loss of a training instance z by an infinitesimally small step ϵ" [2] which leads to the new model parameters like so:

$$\hat{\theta}_{\epsilon,z} = \text{argmin}((1-\epsilon)\frac{1}{n}\sum_{i}^{n} L(z_i,\theta) + \epsilon L(z,\theta))$$

The influence function of the parameters, i.e. the influence of upweighting training instance z on the parameters, can be calculated as follows:

$$I_{up,params}(z) = \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}$$

which is then computed with the Hessian matrix and gradient of the loss function after upweighting instance z - area where we have not much expertise.

The change in parameters is then approximated as:

$$\hat{\theta}_{-z} = \hat{\theta} - I_{up,params}(z)$$

Having found the new parameters (or their approximations), we can then reevaluate predictions on the training set, however, authors in their paper describe a more efficient way, which we however will not present here.

Now, with both change in parameters and predictions, we can do everything described in the Deletion Diagnostics section, but without a need of constant retraining of the model.

The python package implementing this method is the package written by authors themselves, but it does not have proper documentation and was primarily used for their own paper, we believe. We have found much more friendly package called `Influenciae`[11] which we will be using in the practical part to provide a few basic examples practitioners can make use of.

[11] `https://pypi.org/project/Influenciae/`

Chapter 3
Practical Part

3.1 The Disagreement Problem

Disagreement problem is a very vast issue in the field of explainable ML, so in this work we will be primarily focusing on comparing two widely used methods: SHAP and LIME. But before diving into our comparison, let us give a quick summary of these methods from the practical point of view.

3.1.1 Brief (practical) overview of SHAP and LIME methods

KernelSHAP is local model-agnostic method which calculates feature importances for the given data instance by approximating Shapley values. The essential part is that for the given feature Shapley value incorporates all this feature's interactions with other features. Of course, KernelSHAP is not ideal method. Amongst its biggest drawbacks is (1) the fact that it has simplified requirements to the data (such as linear independence) which might lead to incorrect explanations and (2) long computation time in comparison to LIME (example of computation times is presented in Table 3.1 showing significant difference in favor of LIME).

Shapley values can then be used as:

1. local feature importances
2. global feature importances
3. partial dependence plot

All of them are already implemented in the python package `shap`[1] by the authors of SHAP.

LIME is also local model-agnostic method which calculates feature importances for the given data instance by approximating given black-box model with an interpretable model (such as linear regression) in the proximity of the data instance. Feature importances are then feature importances of the approximating interpretable model (in case of Linear models, importances are feature weights w)

LIME authors as well provide python package `lime`[2], supporting tabular, image and text data. In the existing implementation approximating model is also sparse which leads to sparse

[1] `https://shap.readthedocs.io/en/latest/`
[2] `https://github.com/marcotcr/lime`

Method	Seconds	Explained instances	Background instances
KernelSHAP	395.59	137	408
KernelSHAP	110.07	137	100
LIME	4.80	137	408
TreeSHAP	0.03	137	408

Table 3.1 Computation time (in seconds) for KernelSHAP, LIME and TreeSHAP for catboost model and House Price prediction dataset (data explained further on). Seconds - seconds taken to compute explanations for every instance in explained instances. Background instances - number of instances from the training data taken as a background data.

explanations (including only the most important features). Furthermore, number of features to include is hyperparameter, so we as users can select it. However, another hyperparameter *kernel width*, that controls the proximity in which to approximate the model, arguably, makes LIME questionable choice: selecting the wrong value for this parameter leads to incorrect approximations, consequently, explanations, and we as users have no way to see that in the multidimensional space of our data.

3.1.2 Goals of the experiments

Given this brief introduction, these are the questions we will be focusing on regarding disagreements.

- How can we measure how much 2 explanations are different?
  To do this we will be using 6 metrics that measure the difference between 2 feature importance explanations that we described in the theoretical part. The metrics are based on the survey of 25 practitioneers working in/with xAI field and they satisfy very natural requirements one would expect from something that measures the disagreement between two explanations.

- How often do explanations differ?
  To try to answer this question, we have taken two datasets for regression and classification, trained black-box models, calculated local feature importances using SHAP and LIME for each data instance and compared these two sets of explanations using the metrics mentioned above.

  We would also like to test the following hypothesis:

- Given methods of the same nature, we would expect they produce very similar explanations. For example, KernelSHAP and TreeSHAP are methods that approximate Shapley Values, but they do so in different ways. Expectation is that the disagreement between 2 different SHAP explanations would be small, at least much smaller than the difference between explanations produced by the methods of "different nature" (such as SHAP and LIME)

3.1.3 How often do explanations differ?

To answer the second question of the frequency of disagreement, we have decided to (1) take two datasets for regression and classification, (2) train black-box models and (3) calculate local feature importances using SHAP and LIME for each data instance and compare these two sets of explanations using the metrics.

Regression

Data: House price prediction For the regression task, we have chosen easy and interpretable "House Price" prediction dataset from Kaggle[3]. This dataset is suitable for our task because importance of some features is very small (as we will see) and we will make use of it while trying to see if SHAP and LIME can detect that model does not actually use these features.

It has 12 features: house area, number of bedrooms, bathrooms, stories; boolean features whether the house has mainroad, guestroom, basement, hotwaterheating, airconditioning, parking space and 1 categorical feature about the furnishing state of the house: furnished / semi-furnished / not furnished. Some features, such as area, number of bathrooms are quite correlated with the price value of house (coefficient is around 0.50). For the data preprocessing we have only mapped boolean/categorical features to numerical, and imputed few missing values to the most frequent ones, so for the preprocessing we have done the bare-minimum.

Model training: CatBoost For the model performance we have tried to minimize overfitting of the CatBoost model by giving the maximum depth of tree as 2. In the end, the model still overfits, which can be seen by comparing the R2-score on train and test data. Further implementation details could be found in the enclosed notebooks.

Classification

Data: Penguin classification For the classification we also have decided to go with an interpretable dataset: "palmerpenguins" penguin classification data [25] which can be used as an alternative to iris dataset. The goal is to predict which of 3 similar penguin species the animal belongs to based on 6 features: island, bill length, bill depth, flipper length, body mass index and sex. It has 344 records of different penguins in it.

Similarly to the regression data, we didn't perform any data processing apart from encoding categorical features to numerical.

Model training: Random Forest Here, we have selected Random Forest model with the default number of trees: 100. 100 trees may seem as a high number for the dataset of that size because of the high chance of overfitting, however even with 10 trees, train and test accuracies were very close to 1. As a result, we settled for 100 trees with higher chance of overfitting but more stability[4].

Resultant metrics

Regression: Table 3.2 shows the resultant metrics for the regression dataset (CatBoost, trained on House Price prediction data).

We can see that for 1 feature, *feature* and *sign agreements* show 0.226 which means that top feature is the same on average only in one fourth of all the data instances, which is arguably a huge disagreement. Situation improves with $k = 2$, but it stays pretty much the same for the suitable number of top features for this problem: 3-4 which is quite natural for dataset with 12 features. Metrics showing absolute feature ranking agreement – *rank agreement* and *signed rank agreement* – show very little agreement. Metrics using relative feature ranking – *rank correlation* and *pairwise rank agreement* show more agreement and very close results for almost all the k-s.

Classification: Table 3.3 shows metrics for classification data for one label only, Adelie penguin (other 2 show similar picture). We selected this one because this is the most frequent class. In contrast to regression dataset, we see much less disagreement: all metrics except for rank

[3] `https://www.kaggle.com/datasets/yasserh/housing-prices-dataset`

[4] Retraining Random Forest model with fewer trees might result in slightly different forests

k	Feature agr.	Sign agr.	Rank agr.	Signed rank agr.	Rank correlation	Pairwise rank agr.
1	0.226	0.226	0.226	0.226	—	—
2	0.529	0.529	0.168	0.168	1.000	0.409
3	0.567	0.567	0.158	0.158	0.631	0.637
4	0.648	0.648	0.144	0.144	0.565	0.687
5	0.737	0.737	0.145	0.145	0.585	0.708
6	0.786	0.782	0.135	0.134	0.536	0.686
7	0.818	0.808	0.131	0.128	0.580	0.722
8	0.839	0.819	0.125	0.118	0.630	0.747
9	0.859	0.814	0.131	0.114	0.623	0.743
10	0.877	0.821	0.133	0.115	0.664	0.763
11	0.911	0.855	0.144	0.128	0.705	0.781
12	1.000	0.875	0.134	0.117	0.678	0.760

Table 3.2 Disagreement metrics for the SHAP and LIME methods. Model: CatBoost, dataset: House Price prediction.

k	Feature agr.	Sign agr.	Rank agr.	Signed rank agr.	Rank correlation	Pairwise rank agr.
1	0.884	0.884	0.884	0.884	—	—
2	0.820	0.814	0.756	0.756	1.000	0.907
3	0.864	0.833	0.655	0.655	0.884	0.888
4	0.878	0.831	0.564	0.564	0.798	0.864
5	0.874	0.812	0.530	0.530	0.834	0.873
6	1.000	0.866	0.504	0.504	0.819	0.853

Table 3.3 Disagreement metrics for the SHAP and LIME methods. Model: Random Forest, dataset: Penguin Classification, label: Adelie

agreement ones, show agreement coefficients of 0.8 and higher which is very high. The result, however, is probably influenced by the simplicity of the data.

Conclusion

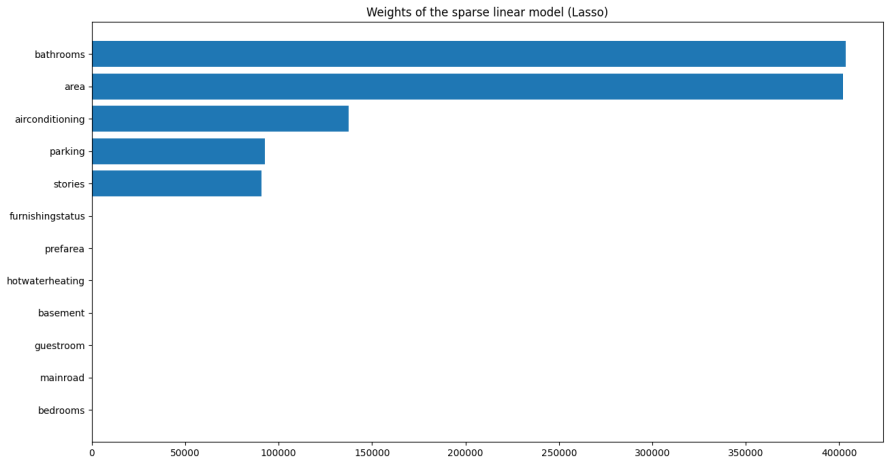
We have compared KernelSHAP and LIME methods on regression and classification data. We have seen that on simple data, such as Penguin classification, these methods show very little disagreement, however, on more complex data - House Price Prediction - the results are less agreeing.

Let us now check how these methods behave with the sparse model.

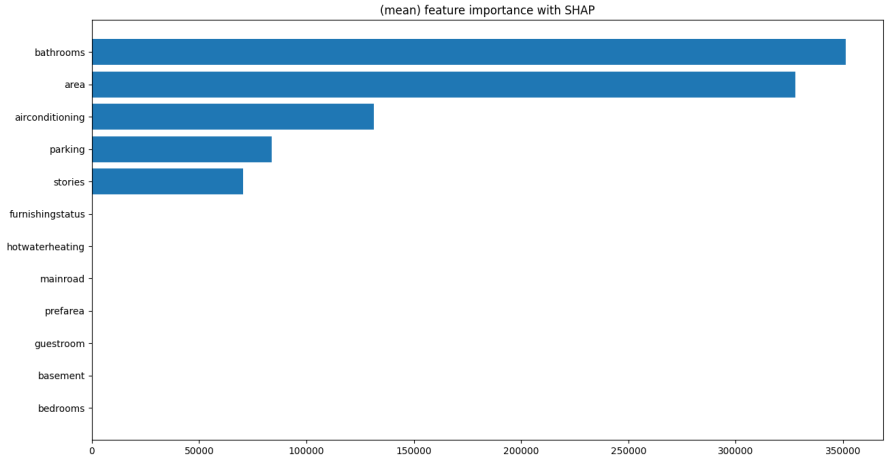
3.1.4 SHAP vs LIME on sparse model

In this part we want to know if SHAP and LIME methods could detect that model does not use features. For this, we have taken the same regression data as in the previous experiment in Section 3.1.3, trained sparse linear model Lasso so that it uses only 5 out of 12 features (the lambda hyperparameter was chosen to use 5 features, so that model's score was as best as possible). For the feature weights of Lasso model to provide meaningful feature importances, we have also standardized data (mean = 0, variance = 1). The goal is to check if KernelSHAP and LIME could detect that this model does not use the rest of the features: if for these not used features importances would be zero or close to it. The resultant Lasso model turned out to underfit the data, but that is the cost of not using all the data available.

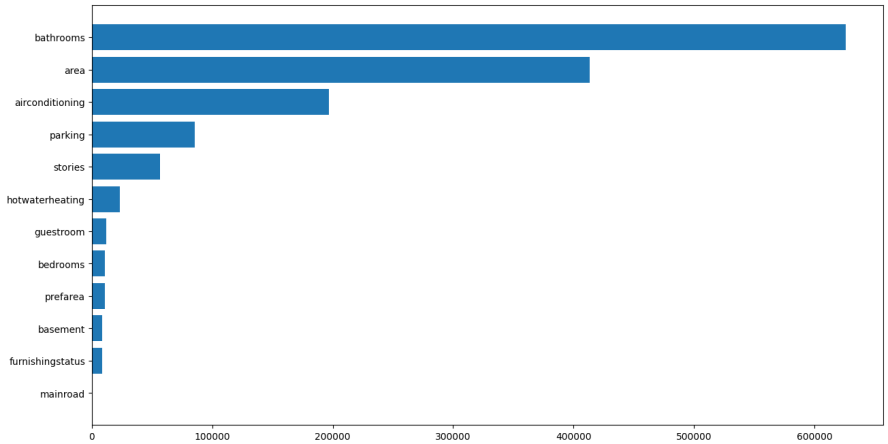
In Figure 3.1 we can see that while real model uses only 5 features: number of bathrooms,



(a) "Real" feature importances: only first 5 features are used. Model's weights are used as feature importances (data is standardized).



(b) SHAP feature importances



(c) LIME feature importances. Feature importance is taken as an average of absolute values of importances over all data, same as SHAP.

Figure 3.1 SHAP vs LIME on sparse model (Lasso)

area, airconditioning, parking and number of stories, SHAP could correctly identify that the rest of the features were not used: their feature importances are zero. On the other hand, LIME still assigns some feature importances to these features, they are, however, rather negligible (especially the last ones), but they are still there. We have to note that since LIME creates a sparse explanation, its implementation has an optional parameter of how many features to have in the explanation. This, however, is not ideal, since for complex models we usually do not know how many features are used by it, and, as we see, while LIME cannot identify it, SHAP can. Furthermore, SHAP feature importances look very similar to the real ones: (1) ordering is the same and (2) even the relative importances are comparable. The similar can be said for LIME as well, considering top-5 features have the same order but relative values seem to be different. We can observe that fact in Figure 3.2.

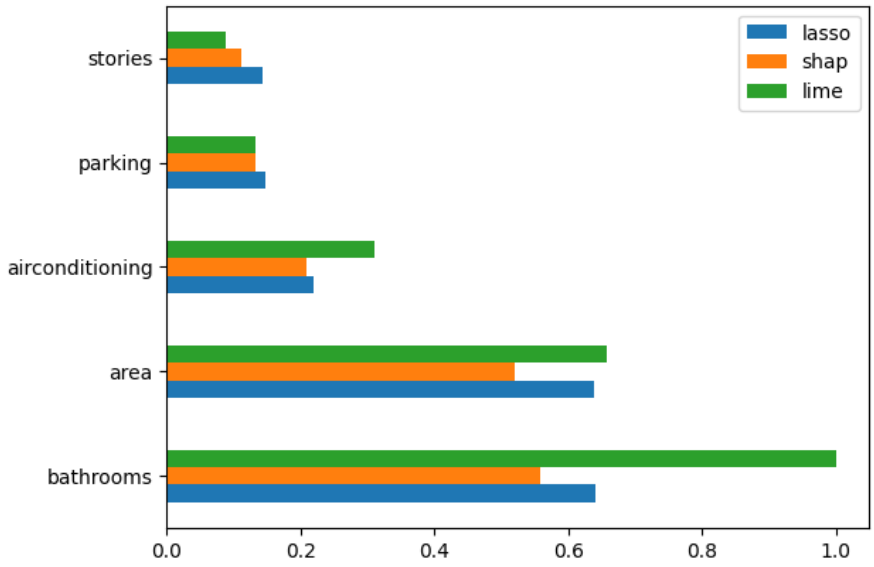


Figure 3.2 Comparing real feature importances with ones generated with SHAP and LIME on a sparse model (Lasso).

3.1.5 Comparing KernelSHAP and TreeSHAP

As already said, now that we know how different explanations SHAP and LIME could give, we would like to verify that 2 methods "of the same nature", model-agnostic KernelSHAP and model-specific TreeSHAP, provide similar results. Just like before, we will be taking two datasets: "House price prediction" for regression and "palmerpenguins" for classification, two same models trained on these data and comparing average metrics between KernelSHAP and TreeSHAP on the datasets. To remind of the results from before, we have got that on the regression data KernelSHAP and LIME were giving comparatively different results and on classification the results were much more similar. Although it is worth mentioning that the classification dataset is more simple data having only half as many features as regression and model trained on it has 100% accuracy even on the test data.

Our hypothesis was that explanations provided with KernelSHAP and TreeSHAP would be almost alike: having high agreement metrics on both datasets. We have computed the metrics using KernelSHAP and TreeSHAP, and they are displayed in Table 3.4.

As we can see, the average *feature* and *sign agreements* are very high starting from 0.87 for 1 feature, 0.9 for 2, 0.917 for 3 and getting higher and higher with each new feature added. Metrics

k	Feature agr.	Sign agr.	Rank agr.	Signed rank agr.	Rank correlation	Pairwise rank agr.
1	0.869	0.869	0.869	0.869	—	—
2	0.898	0.898	0.781	0.781	1.000	0.891
3	0.917	0.917	0.735	0.735	0.909	0.927
4	0.932	0.932	0.703	0.703	0.907	0.927
5	0.962	0.962	0.699	0.699	0.902	0.922
6	0.979	0.979	0.713	0.713	0.928	0.936
7	0.974	0.974	0.721	0.721	0.929	0.933
8	0.960	0.960	0.713	0.713	0.948	0.948
9	0.951	0.951	0.697	0.697	0.958	0.953
10	0.962	0.962	0.674	0.674	0.961	0.954
11	0.995	0.995	0.668	0.668	0.969	0.960
12	1.000	0.999	0.692	0.692	0.968	0.957

Table 3.4 Disagreement metrics for KernelSHAP and TreeSHAP methods. Model: CatBoost, dataset: House Price prediction.

k	Feature agr.	Sign agr.	Rank agr.	Signed rank agr.	Rank correlation	Pairwise rank agr.
1	0.860	0.860	0.860	0.860	—	—
2	0.826	0.826	0.686	0.686	1.000	0.977
3	0.868	0.868	0.655	0.655	0.785	0.853
4	0.916	0.913	0.605	0.605	0.758	0.855
5	1.000	0.958	0.616	0.616	0.808	0.878
6	1.000	0.965	0.680	0.680	0.890	0.919

Table 3.5 Disagreement metrics for KernelSHAP and TreeSHAP methods. Model: Random Forest, dataset: Penguin classification, label: Adelie

that compare relative ordering - *rank correlation* and *pairwise rank agreement* - also show very high average agreement between two explainability methods. If we compare this table with the one that shows disagreement measures between KernelSHAP and LIME, we can see, especially for low k-s, that disagreement metrics on the similar methods - in our case, KernelSHAP and TreeSHAP - show much fewer differences on average than on SHAP and LIME.

In case of penguin classification for the same penguin class Adelie, metrics in Table 3.5 surprisingly show similar results as before - some values are higher, some lower, but, generally, they look alike. On the other hand, for Gentoo penguin we got perfect agreement scores for all the metrics for all k-s which we didn't have before.

Overall, we can say that at least on these data KernelSHAP and TreeSHAP methods output much more similar results, as measured with metrics from the Disagreement paper, than KernelSHAP and LIME methods.

3.2 Practical impact of correlation on different methods

As already stated in the theoretical part, in the practical part for correlation we will be comparing marginal and conditional versions of different explainability methods on correlated and not correlated features.

First things first, let us take a look at how different plots method are dealing with correlation - namely PDP, M Plot and ALE-plot.

3.2.1 PDP, M Plot and ALE-Plot

For this task we have generated artificial data with 3 features - first and second features x_1, x_2 being strongly correlated with correlation coefficient $r = -0.94$ and third feature x_3 being independent. Then we sampled 2000 (1500 training and 500 testing) instances from joint distribution described by this function:

$$f(x) = ax_1 + bx_2 + cx_1x_2 + dx_3 + \epsilon$$

where $a = -4, b = -15, c = -16, d = 2$ are coefficient randomly chosen in range from -20 to 20 and ϵ is random error that follows standard distribution with mean 0. Note: we have taken x_1x_2 term to see what dependence different plots will show for linear model, where the dependency should always be linear, regardless whether feature is correlated or not.

Then we trained 2 models: Random Forest and Linear Regression without any tuning because training perfect model is not our goal. Since our function has multiplication term, Linear Regression could not have fit data perfectly and has quite big train and test errors, but this is of little importance in our task. Random Forest was fit quite well even with default hyperparameters.

Plot methods implementation

PDP First and the easiest to both understand and implement is Partial Dependence Plot which plots marginal effect of this feature on the target variable. PDP is already implemented in a number of python libraries such as `scikit-learn` [10], but because we have not found any implementation of M Plot which would be very close to that of PDP, we will be implementing PDP as well as M Plot.

To implement PDP, we need to:

1. Define the grid. The "default" option is to split the feature domain into n equally sized grid cells. Knowing, however, that most features are not uniformly distributed (leading to some cells containing too few instances), we will define our grid by the quantiles: this way, every grid cell will have the same number of feature values $n/\#\text{cells}$
2. Then for every grid value, calculate mean prediction assuming that given feature in all data instances have this grid value.
3. Finally, plot resultant mean predictions against grid values.

As we have already mentioned, the problem is that if feature of interest is correlated with other(s), PDP will compute the effect over unrealistic data points. This might be advantageous if you are following "True to the model" approach, but definitely not beneficial for "True to the data".

M Plot The only difference between M Plot and PDP is that instead of assuming that given feature is independent from all others and therefore using feature's marginal distribution, M Plot calculates effects using feature's conditional distribution.

To implement M Plot, we need:

1. Define the grid. Here we are following the same quantile approach as in PDP (and ALE).
2. Then for every grid value, get k instances which have closest feature values to the grid value. On this sample, calculate mean prediction assuming that given feature in all data instances have this grid value. Getting k instances with closest feature values can be done with "k-nearest neighbors" technique. Figure 3.3 shows the scatter plot between 2 correlated features and how we can select k closest data points for $k = 200, 500, 1000$.
3. Plot resultant mean predictions against grid values.

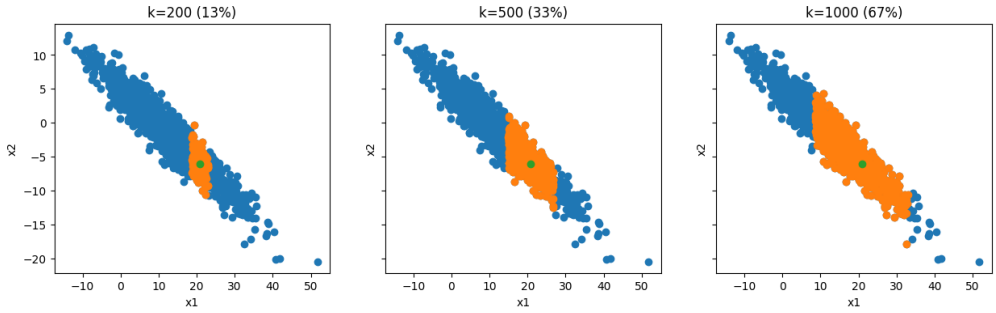


Figure 3.3 Approximating feature's conditional distribution with k-nearest neighbors method when features x_1 and x_2 have a strong collinearity. The bigger the k, the more instances are being taken to compute the effect of x_1 onto target variable.

The drawback of this implementation is that for each grid value we do the approximation on only k samples, which can also be misleading for small data and/or small k. Furthermore, k is hyperparameter which we need to select beforehand. In our examples, we are using 20% of training data which is 300 out of 1500.

For both our PDP and M Plot implementations we also center plot around average prediction which is not done in existing implementations but this will be of help for comparing results of our plots and ALE, since that's what ALE plot does.

ALE-Plot For this plot, we are using python `alibi` package [26]. It is important to mention that its ALE implementation also uses quantile approach, just like we do for our PDP and M Plot.

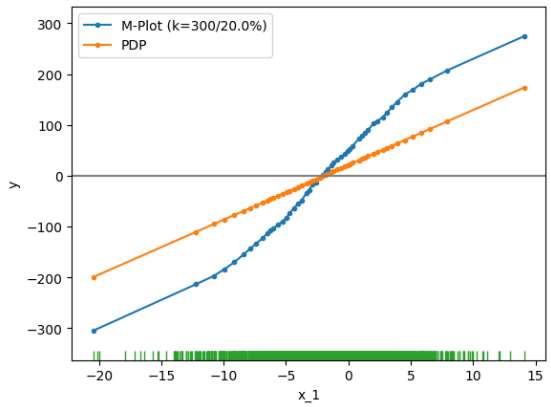
Results on Linear Model

Correlated feature The results are shown in Figure 3.4. Plots on the left show plots for the first feature x_2 which is highly correlated with feature x_1. The top one shows 2 plots: M Plot and PDP. The bottom one shows ALE-plot. Comparing M Plot and PDP we can see that both show exactly straight line which is "correct" according to "True to the model" approach, but not "True to the Data": since model used is Linear Regression, every single feature by itself always has linear relationship with the target variable, while we know that in reality is dependent not only on the linear term bx_2, but also non-linear term cx_1x_2. In this regard, M Plot does a better job at showing the non-linear nature of target variable dependence on a feature by showing a curve line. We can see, however, that maximum feature effect that is shown by ALE and PDP are different from those shown by M Plot: the former showing effect from -200 to 150 while latter from -300 to 300. This tells us that not only PDP/ALE and M Plot show the relationship of different nature (linear vs non-linear), but also provide quantitatively different results.

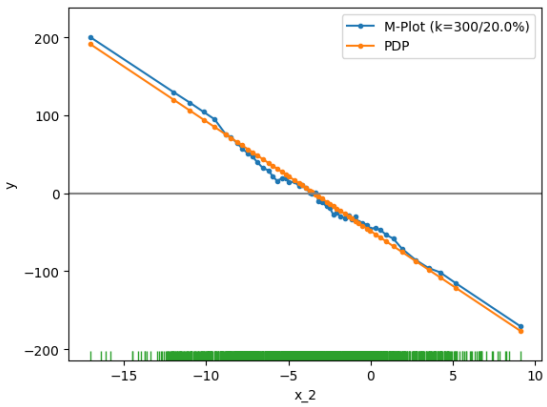
Not correlated feature In figure 3.4 plots on the right show the same plots - PDP and M Plot on top and ALE at the bottom - but for not correlated feature now. Here, however, we do not observe any disagreements: all three plots show similar linear relationship for linear model for linear feature which in advance is independent. Since this is what we would have expected, these plots show no surprises.

Results on non-Linear Model

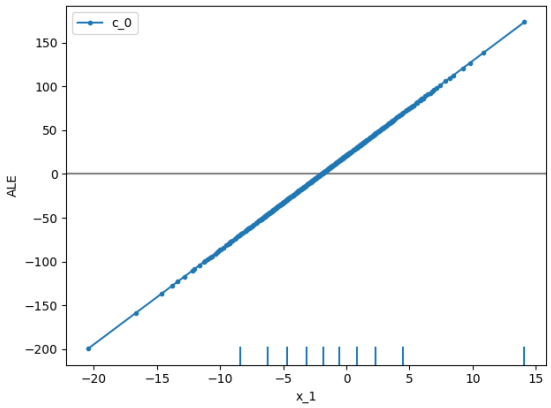
Correlated feature The results for non-linear model (in our case, Random Forest) are shown in Figure 3.5. Just like before, plots on the left show plots for the first feature x_2 which is highly



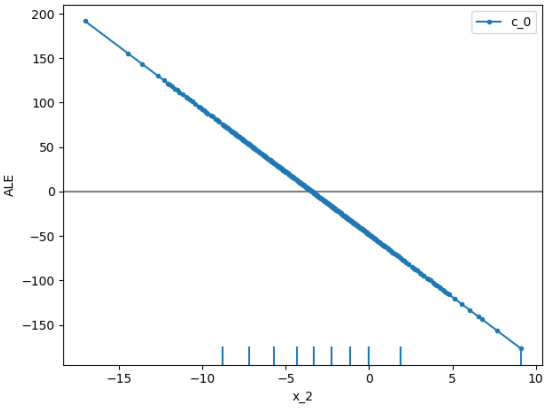
(a) M Plot and PDP for correlated feature and Linear Model



(b) M Plot and PDP for not correlated feature and Linear Model



(c) ALE plot for correlated feature and Linear Model



(d) ALE plot for not correlated feature and Linear Model

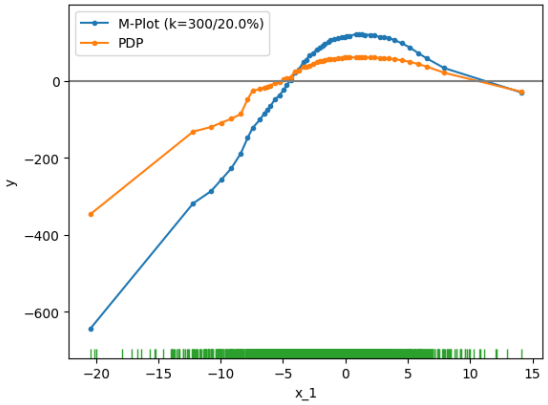
Figure 3.4 Results for ALE plot, M Plot and PDP on correlated and not correlated features on Linear Model. (note: on the top, rugplot is plotted in green)

correlated with feature x_1. This time, we see that all 3 plots show similar, yet not same picture. On one hand, they all reveal that feature effect is parabolic and at first rises and then decreases. On the other hand, top part of PDP (above the mean prediction) is much more like to ALE, as for the lower part M Plot shows more resemblance to ALE. In other words, when showing positive effect of feature, M Plot overestimates it (comparing to ALE) and PDP does not, while for negative feature effect PDP underestimates it (again, comparing to ALE) and M Plot does not.

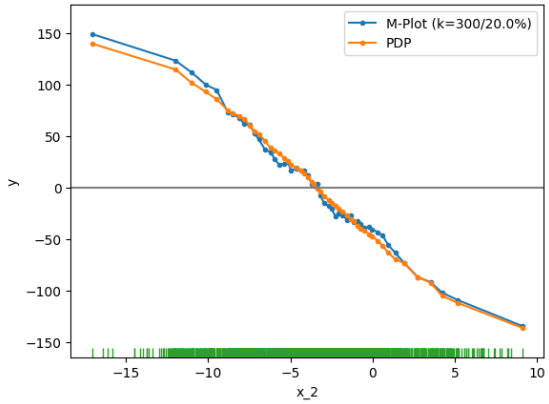
Not correlated feature Looking at not correlated feature with linear effect on the target value, and considering that our random forest model fit the training data well, it is, again, of no surprise that our plots show this feature's linear effect and all of them showing the closely same plot. The only small caveat here is that locally we can see some disturbances along the curve, but arguably this is not so important.

3.2.2 KernelSHAP and TreeSHAP

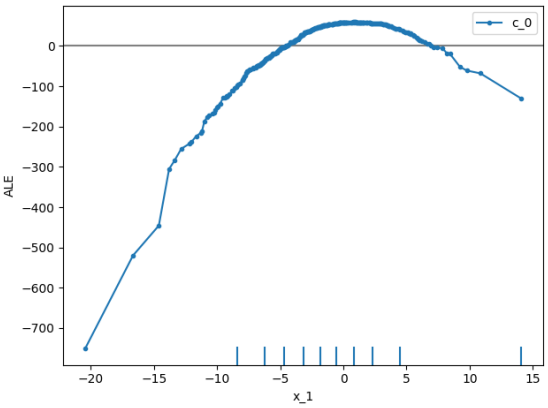
KernelSHAP and TreeSHAP are both SHAP methods that are used to approximate Shapley values. KernelShap is a model-agnostic method utilizing marginal distribution while TreeSHAP



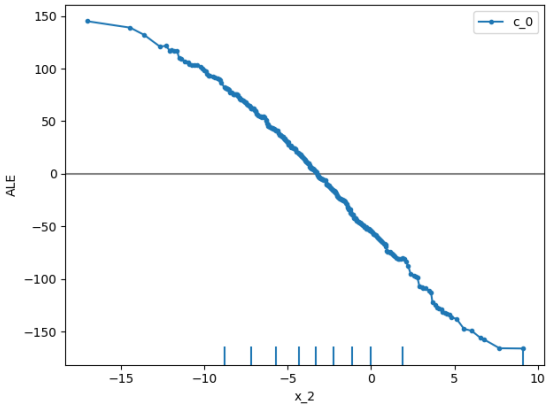
(a) M Plot and PDP for correlated feature and Random Forest



(b) M Plot and PDP for correlated feature and Random Forest



(c) ALE plot for not correlated feature and Random Forest



(d) ALE plot for not correlated feature and Random Forest

Figure 3.5 Results for ALE plot, M Plot and PDP on correlated and not correlated features on Non-Linear Model (Random Forest). (note: on the top, rugplot is plotted in green)

is a model-specific to tree models that can use both marginal and conditional feature distributions depending on the `feature_perturbation` parameter. "`interventional`" one, essentially, utilizes marginal distribution and "`tree_path_dependent`" - conditional. Another way to view these versions is "True to the model" and "True to the data" respectively [27].

Our goal is to compare the results of these 3 methods with the results of exact Shapley values on different correlated data to see which method does the best approximation. For computing all 4 Shapley values we were using the python `shap` package that was written by the creators of SHAP methods.

Practically, our procedure was as follows:

1. We have taken 3 different datasets that do have correlated features. We describe each dataset and its results individually in the following subsections.

2. Split them into training and testing sets.

3. Trained a RandomForest model on training data and calculated error metrics (MAE for regression and accuracy for classification) to check that model at least learnt something (perfect model is not a goal for us). We took RandomForest because TreeShap methods are model-specific to tree methods, and RandomForest is widely known. In the end, it could have

been any tree method, e.g. CatBoost, XGBoost, results on a concrete model do not interest us.

4. Sampled data from the training set that we will be further using as a background data for computing SHAP values on.

5. Computed exact SHAP values, kernel SHAP values and tree SHAP values with 2 different parameters of `feature_perturbation` on the test data using the background data sample. We chose test data to compute Shapley values on, because test data are more likely to be used for explanations in the real scenario.

6. And, finally, compared 3 resultant Shapley values approximation methods with exact Shapley values to see the difference for single methods with correlated data.

For the last point, to compare the results between exact Shapley values and its approximations, we calculated mean absolute error between exact values and approximations across all datapoints and all features (also per each class in case of classification) like so:

$$\text{Error} = \frac{1}{N \cdot M} \Sigma_{i=1}^{N} \Sigma_{j=1}^{M} |\text{shap\_approximation}_{i,j} - \text{shap\_exact}_{i,j}| \tag{3.1}$$

where N is the number of data instances in testing subset and M is the number of features in the dataset.

Now, let us describe results for each dataset.

Diabetes dataset (regression)

The first dataset we have taken was diabetes dataset from `scikit-learn` library [10].

It contains data about around 400 individuals each having 10 features: age in years, sex, body mass index (BMI), blood pressure (bp) and 6 others describing blood parameters (features s1-s6). Each feature was standard scaled. Target variable is a "quantitative measure of disease progression one year after baseline".

Figure 3.6 shows the correlation matrix for this dataset.

We have trained Random Forest model, for each of the SHAP method chosen computed SHAP values on testing subset and then calculated the error as described in the equation 3.1.

The results are shown in Figure 3.7. We can see that KernelSHAP has an error of approximately 1.3 and TreeSHAP with `tree_path_dependent` around 1.73, while TreeSHAP with `interventional` shows an error of only 0.078 which is a whole order less than those of 2 previous SHAP methods. We also see that "SHAP base value" is around 150, which is by SHAP definition an approximation of mean prediction for testing data (since we are using a testing subset for explanations). So even the biggest error of TreeSHAP method with `tree_path_dependent` is only around 1%, which is arguably not a big one. In fact, the feature importances given by these SHAP methods are alike, sometimes misplacing features with similar importances. Another observation we can make is that, surprisingly, KernelSHAP method (marginal) gives a similar error as TreeSHAP with `tree_path_dependent` (conditional), while KernelSHAP with `interventional` - also being marginal - gives a lesser error.

However, when deciding which method approximates exact Shapley values more when dealing with correlated data, TreeSHAP with `interventional` does a much better job, at least for this data. Let us take a look at other datasets to see if our results also apply to them.

Avocado prices (regression)

This dataset describes price levels for avocados. We have found it on Kaggle [5] and here is the description of it from the original source:

[5] `https://www.kaggle.com/datasets/neuromusic/avocado-prices`

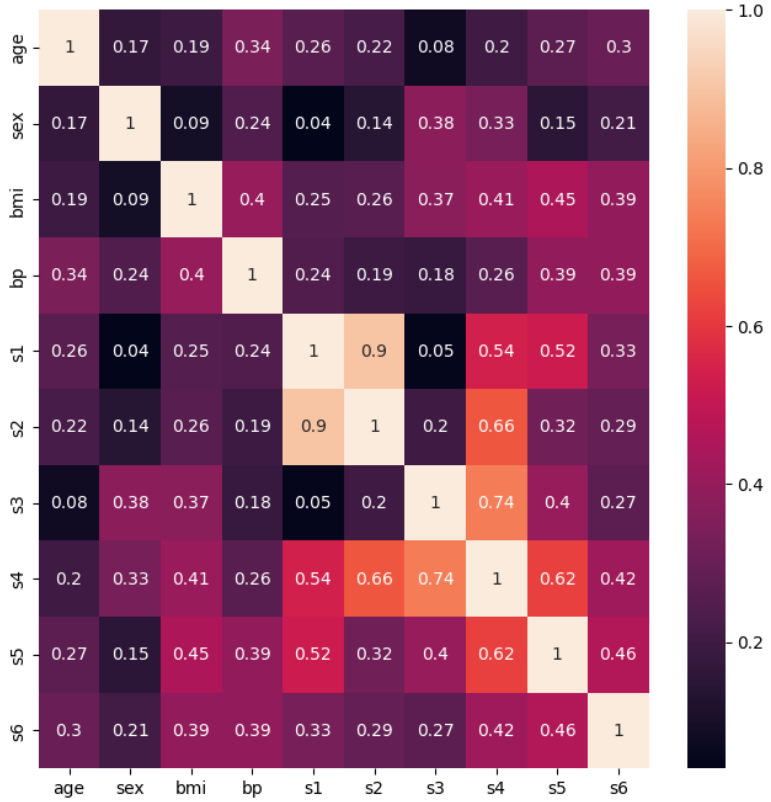


Figure 3.6 Matrix of absolute values of correlations for diabetes dataset

> The table below represents weekly 2018 retail scan data for National retail volume (units) and price. Retail scan data comes directly from retailers' cash registers based on actual retail sales of Hass avocados. Starting in 2013, the table below reflects an expanded, multi-outlet retail data set. Multi-outlet reporting includes an aggregation of the following channels: grocery, mass, club, drug, dollar and military. The Average Price (of avocados) in the table reflects a per unit (per avocado) cost, even when multiple units (avocados) are sold in bags. The Product Lookup codes (PLU's) in the table are only for Hass avocados. Other varieties of avocados (e.g. greenskins) are not included in this table.

It contains around 18000 instances and 12 features. We, however, sampled only 1000 of instances in our work to speed up the computations for Kernel and Exact SHAP methods. The correlation matrix is shown in Figure 3.8. We can see that many features are correlated which is exactly what we need.

For this dataset, we did exactly the same procedure as we did with diabetes data - trained RandomForest model on it and computed the error between different marginal and conditional SHAP methods and exact Shapley values - and the results for the avocado prices dataset are shown in Figure 3.9.

We see the similar picture as before. The biggest error comes from KernelSHAP with `tree_path_dependent` method closely followed by KernelSHAP being only twice less and TreeSHAP with `interventional` having an extremely small approximation error (around 10^{-6}). Next, comparing to the base value, errors themselves are not very high: first and second being close to 1%, while third one, as we already noted, being negligibly small.

So far, the results of our experiment still hold: TreeSHAP method with `interventional` is a much better approximator of SHAP values when it comes to the correlated data than

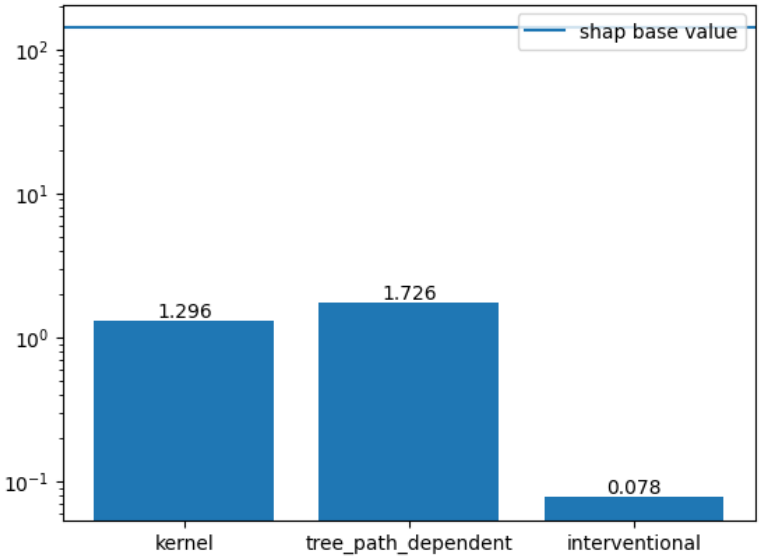


Figure 3.7 Comparing SHAP values generated by KernelSHAP (first), TreeSHAP with `tree_path_dependent`, TreeSHAP with `interventional` with real Shapley values on the diabetes dataset (regression). The values in the plot are average absolute errors over all features and all data points in the testing set. SHAP base value is average target value over testing set.

KernelSHAP or TreeSHAP with `tree_path_dependent`.

Let us now proceed to our last, classification, dataset and see if results we have seen also apply to it.

Students performance classification

This dataset also comes from Kaggle[6] with the original source from a learning management system that provides access to educational resources and has an ability to track students' experience.

The goal is to predict what performance level student achieves - low, middle or high - based on 15 different features. Those consist of 3 major categories: (1) demographic such as age or gender, (2) academic background such as educational stage, grade and section, (3) behavioral such as number of raised hands on classes, opening resources, answering survey by parents, and school satisfaction.

Correlation matrix for this dataset is shown in Figure 3.10. We can see again that it has a number of highly correlated features, such as Nationality and Place of Birth, or Number of Raised Hands and Number of Visited Resources.

Let us now see the results of our experiment on this data in Figure 3.11. We can, once again, see the similar picture with the SHAP values of KernelSHAP and TreeSHAP with `tree_path_dependent` showing almost the same with each other and much higher difference than TreeSHAP with `interventional`, which holds for each class.

Conclusion

We have taken 3 correlated datasets - 2 for regression and 1 for classification, for each of them calculated mean absolute error between 3 SHAP approximations methods - KernelSHAP (marginal method), TreeSHAP with `tree_path_dependent` and TreeSHAP with `interventional` - and found that the latter was approximating exact Shapley values notably better, almost perfectly.

[6] `https://www.kaggle.com/datasets/aljarah/xAPI-Edu-Data/data`

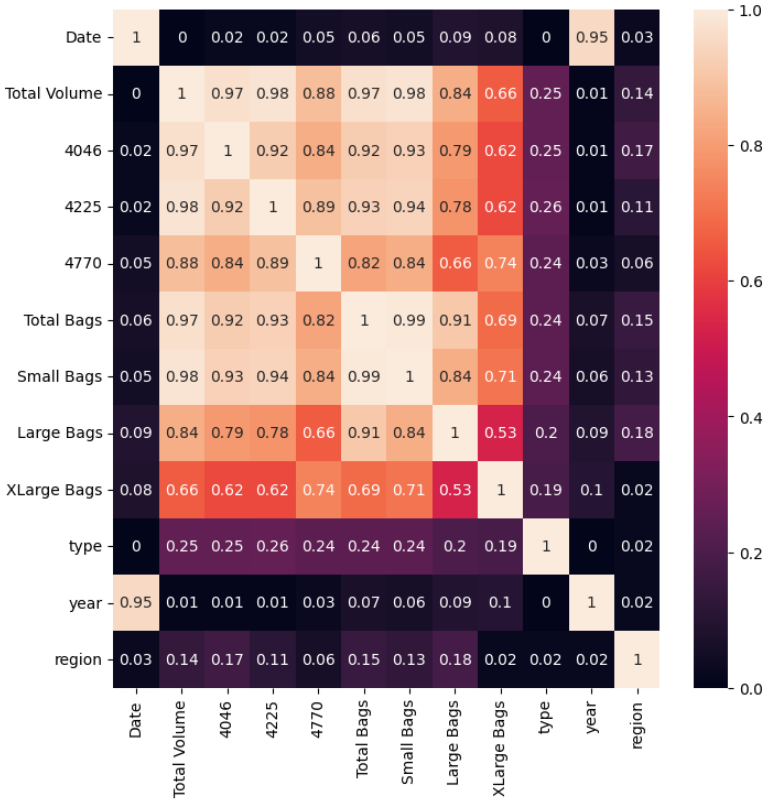


Figure 3.8 Matrix of absolute values of correlations for avocado prices dataset

Furthermore, we have seen that first two methods, even though one is marginal and other is conditional, give similar results with regards to the mean absolute error, which was surprising.

We have to note however, that even though marginal TreeSHAP with `interventional` parameter approximates exact Shapley values very well, it does not mean that it is the method to choose when dealing with correlated data. One still has to keep in mind the problem of "True to the data" and "True to the model" and act accordingly, that is by knowing the goal of your explanation. And for more general purpose do not forget about the "recipe" on how to handle correlation which was summarized already in the theoretical part in Section 2.9.4

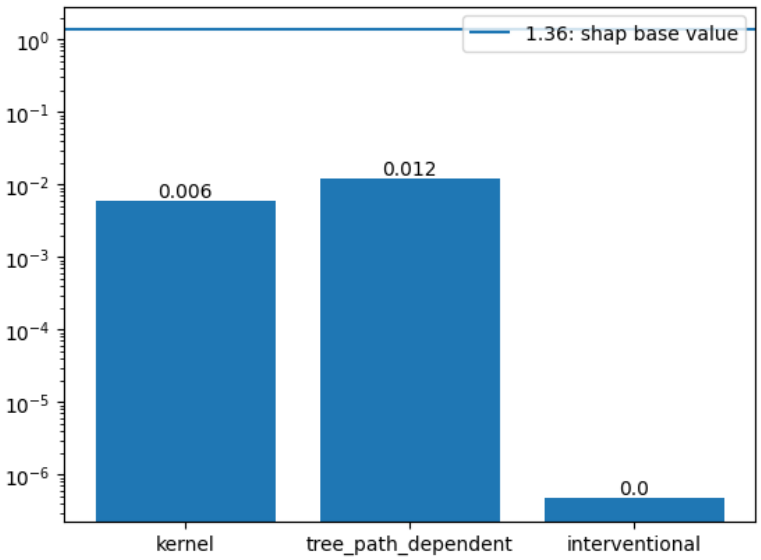


Figure 3.9 Comparing SHAP values generated by KernelSHAP (first), TreeSHAP with `tree_path_dependent`, TreeSHAP with `interventional` with real Shapley values on the avocado dataset (regression). The values in the plot are average absolute errors over all features and all data points in the testing set. SHAP base value is average target value over testing set.

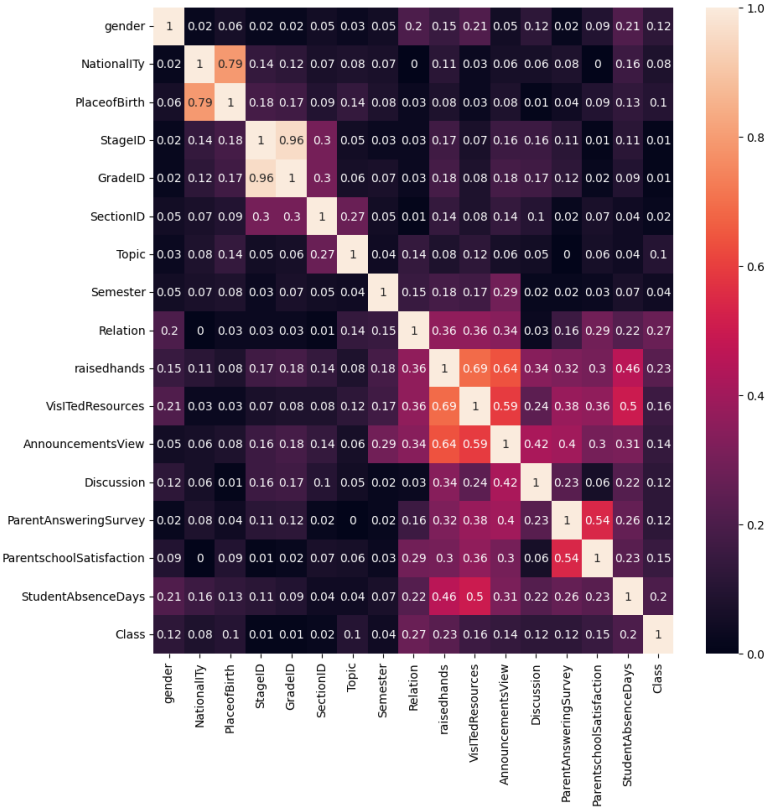


Figure 3.10 Matrix of absolute values of correlations for student performance classification dataset

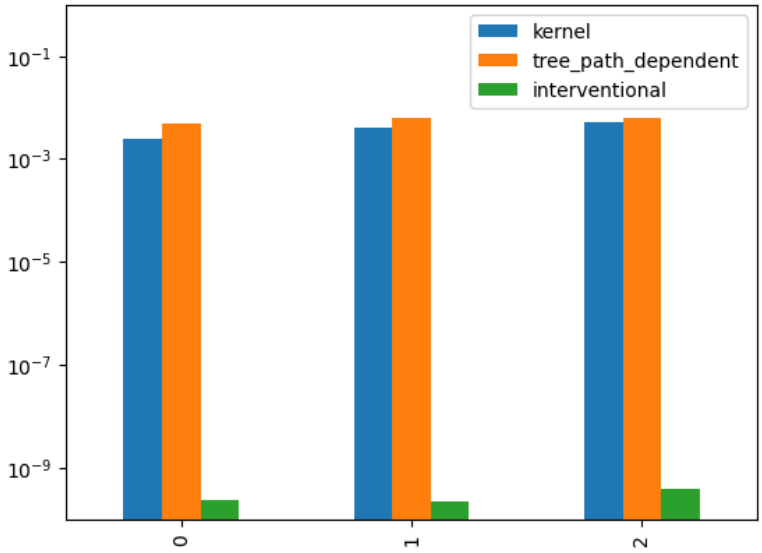


Figure 3.11 Comparing SHAP values generated by KernelSHAP (first), TreeSHAP with `tree_path_dependent`, TreeSHAP with `interventional` with exact Shapley values on the student performance dataset (classification). The values in the plot are average absolute errors over all features and all data points in the testing set per each label.

3.3 Practical examples for Neural Network methods

In this section, we will be showing the practical usages of the Saliency Maps and Influential Functions methods on the simple MNIST digits dataset [28].

3.3.1 Data overview and model training

MNIST digits is a classification dataset of handwritten digits from 0 to 9. It has 60000 training instances and 10000 test instances. Each image has a dimension 28x28. This dataset is often used for example and benchmark purposes.

On it we have trained a simple Convolutional Neural Network that has 2 convolutional layers with 32 and 64 filters with the kernel size 5 and ReLU activation function. Between convolutional layers we also used max pooling layers, and the we also added a Dense layer with 128 neurons before the final Dense layer with 10 neurons for each of the digit class with Softmax activation function. The model structure was found on Kaggle[7].

We have compiled and fitted the model for 10 epochs and resultant test accuracy was 99.3%.

To conduct our further model analysis with the help of explanation methods we described we needed to find some interesting instances. For this, we have decided to take several test instances which the model was misclassifying, preferably with high score.

We have found that among 10000 test images, only 72 were misclassified. The left table in Table 3.6 shows how often the model was misclassifying particular digit and the left one shows with what digits the model was confusing the digit 9 - the most often misclassified number. We can see that from 72 images, the most frequently misclassified digit was 9 with 17 instances followed by 6 with 10. For 10 out of 17 9s the model was predicting 4 followed by 3 and 3 for 5 and 7 respectively.

The most important conclusion we can make at this point is that our model is relatively often confusing nines with fours. Let us now investigate why that is.

Digit	# misclassifications
9	17
6	10
2	9
5	9
8	9
3	8
7	5
0	2
1	2
4	1

9 classified as:	#
4	10
5	3
7	3
0	1

Table 3.6 How the model misclassifies nines over the test set.

3.3.2 Saliency Maps

To proceed with our analysis we have found one 9 digit image from the testing set that the model incorrectly classified as 4 with very high score. Then we have found a 9 that was perfectly classified by the model and a 4 that resembles our first 9 and that the model also perfectly classified.

[7] https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist/notebook

Now, we would like to check why the model classifies one 9 correctly and the other one as 4. First method we will be using is Saliency Maps which shows us what pixels the model found the most relevant for classification of a given class. Each gradient can be negative or positive number which is why usually to see the relevance of the pixel the absolute value of its gradient is taken. Since, however, Vanilla Gradient method tends to scatter the gradients among more pixels, instead of taking an absolute value we can also square them to see the most relevant ones, which is what we did for our 3 instances. The package we have been using is `innvestigate` [8] by [29], although there is another package `keras-vis`[9] that has support of Saliency Maps and other gradient-based methods. The results are in Figure 3.12

The top row (a) shows the image of 9 that was incorrectly classified as 4 with very high score of 0.91, Vanilla Gradient with absolute value of gradients image showing the most relevant pixels for the prediction, and Vanilla Gradient with gradient values squared. The middle row (b) shows a perfect 9 on the left and the outputs of Vanilla Gradient to the right. And the bottom row (c) shows a perfect 4 on the left and the outputs of Vanilla Gradient to the right.

First, we can note that correctly classified 9 indeed looks like a 9, while the one classified as a 4 does in fact resemble a 4, but still looks like a 9. From what Saliency Maps method shows us, it is apparent that for the 9 classified as 4, model primarily takes into account the circular part of the image and, most importantly, the part where the vertical slopes do not connect into the loop like they do for a 9 but not for a 4. We also see a somewhat similar picture for the correctly classified 4 with the gradients having higher values. In contrast for a "real" 9 the model finds the lower part of the loop more relevant and, in turn, gives less importance to the upper part.

Thus, we can see for this particular image of a 9, that the model incorrectly classifies as a 4 with high score, the model does in fact concentrate on the patterns that it used for the classification of a 4 and not a 9. We would argue that this is a valuable information to get from the analysis for the particular image.

The limitation of Vanilla Gradient is that it is very local in the sense that the explanation it provides is only one image. To compare it with other instances, we first needed to manually find the image of a perfect 9 and a perfect 4 that resembled the image being analyzed. The other method - Influential Instances - while being both local and global can also give more insights into the model for local explanations. Let us now see how.

3.3.3 Influential Instances

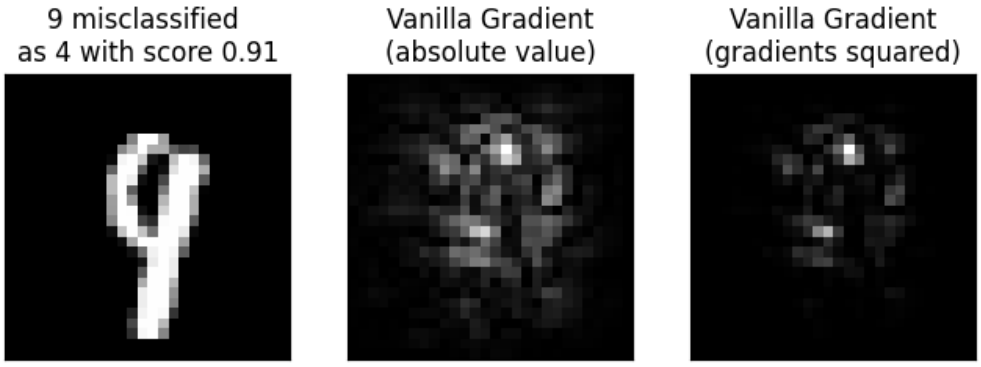
Local explanations

First, we will finish off our example with misclassified image with number 9. The idea behind Influential Instances, or more specifically Influential Functions, for local explanations is that, given an instance of interest, it can measure the influence of every instance in the training set to this instance. It does so by finding an approximate answer to the question "How would the prediction for instance z change if we removed the instance z_i from our training dataset?" As already noted, Influential Functions does not really remove the instance from the data and then retrain the model from scratch, it simulates it by upweighting the loss of the removed instance and then approximating the prediction change with the help of the gradients. The bigger the change, the bigger the influence. Naturally, we can sort all the instances by their influence and find the top-k most influential instances for the given instance.

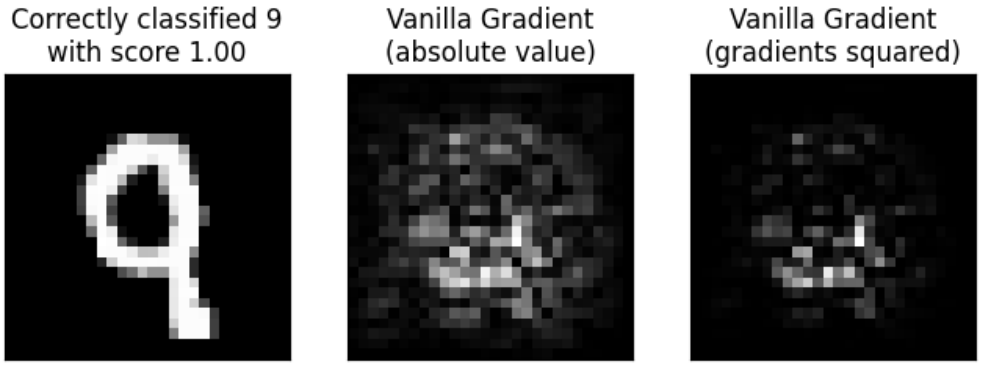
For this we have found a `Influenciae` python package[10] that can do the job.

So, Figure 3.13 presents top-25 most influential instances for our 9 from the training set sorted from the most to the least influential.

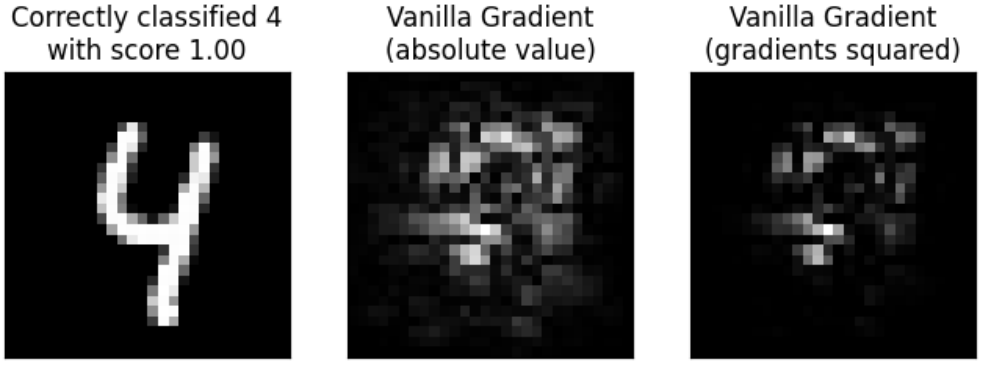
[8] `https://pypi.org/project/innvestigate/`
[9] `https://raghakot.github.io/keras-vis/`
[10] `https://pypi.org/project/Influenciae/`



(a) 9 digit from the test set misclassified as 4 with high score



(b) 9 digit from the test set correctly classified with the perfect score



(c) 4 digit from the test set correctly classified with the perfect score

Figure 3.12 Comparing Vanilla Gradient output on 3 images: 9 incorrectly classified as 4 (top row), perfectly classified 9 (middle row), and perfectly classified 4 (bottom row). The left column shows original images, middle - Vanilla Gradient with absolute gradients, right - Vanilla Gradient with squared gradients.

As we can see among 25 most influential images for our misclassified nine first 15 are also nines that the model misclassifies as fours. We, as humans, can also note that these nines also look similar to the our nine: they all don't have a finished loop at the top which makes it resemble a four. Moreover, Table 3.7 shows with that digits our model misclassifies nines with in the training set, and we can see that from 27 misclassified images, 21 were wrongly classified as fours.

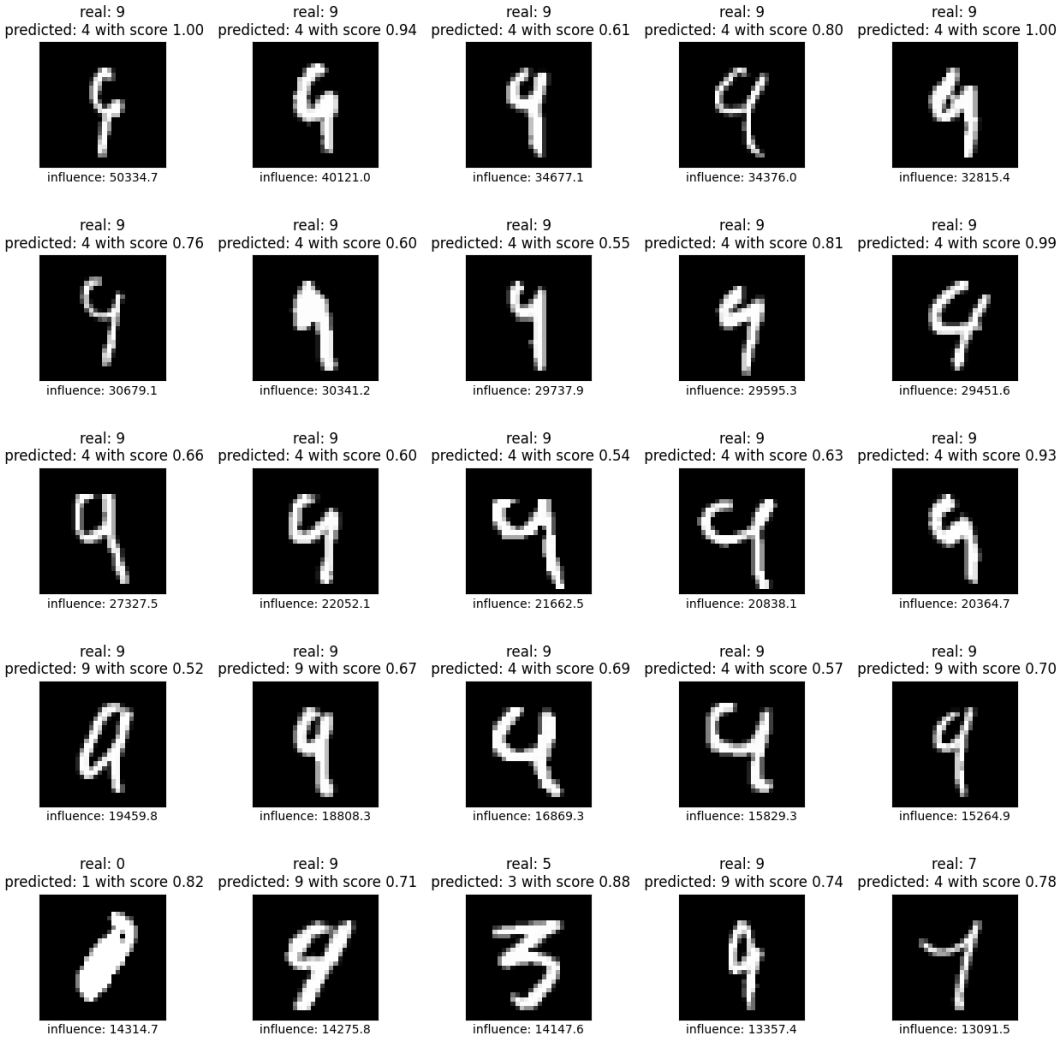


Figure 3.13 Top-25 most influential images from the training set on the incorrectly classified 9 as 4. First 15 are also nines misclassified as fours.

9 classified as:	#
4	21
7	4
8	1
3	1

Table 3.7 How the model misclassifies nines over the training set.

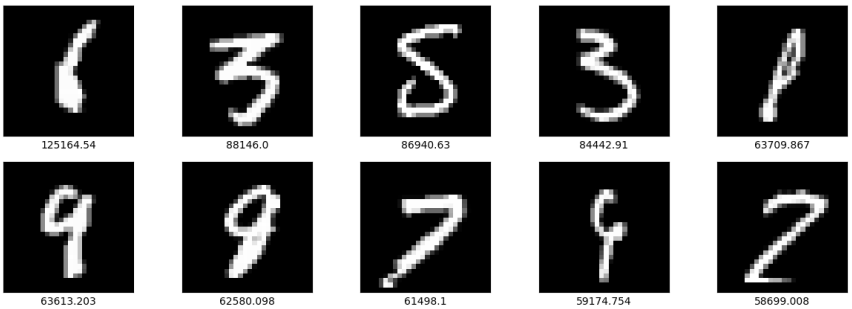
This is a valuable insight of our model and data. Having found this kind of bias, we can now decide how to deal with it. More importantly, we have found it by the means of explainable AI. It is open to debate whether it would be possible to do the same without it, however it would most definitely take more time and effort to do it on the more complex data and network model.

Global explanations

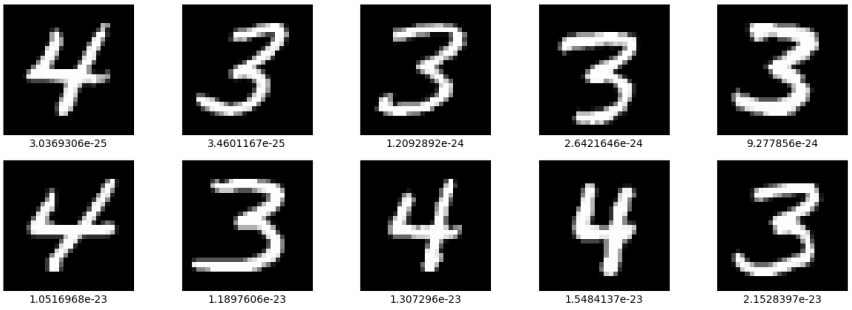
Having found the reason why model was misclassifying nine, now we would like to show the Influential Instances method used for the global explanation. The idea is similar to local explanations, except that now instead of measuring the influence on a another instance, we are measuring the influence on the model. In case of neural networks, we measure the change in the model parameters after simulating deletion of the instance.

The result could be the top-k either most or least influential training instances on the model. The former could be useful to know what training instances impacted the model parameters most, the latter - to find out what training instances had the least impact on the model parameters. Moreover, if their influence is 0 or close to zero, this means that removing them from the training set should not change the parameters of the model, hence its performance, which could be valuable if one wants to find training instances that bear no useful information to the model, for example, to reduce the training set size to further reduce the time of subsequent model (re)trainings.

We will now show top 10 most and least influential images on our model in Figure 3.3.3.



(a) The 10 most influential training instances overall.



(b) The 10 least influential training instances overall.

Figure 3.14 Top 10 most and least influential instances on the model. Values under images are influences calculated by the Influential Functions method.

We can see that while the most influential instances look very specific and unique, the least influential instances are just repetitive images, removing which should not change model's parameters and, hence, its performance. This might be not the most useful insight for the model trained on the MNIST dataset, but it might be much more meaningful for the more complex data.

Chapter 4

Recommendations

In this chapter, we would like to make a brief summary of what we described in theoretical and practical parts of our work and present some recommendations when dealing with inconsistent explainability outputs.

Correlated Features

One of the first things to check when faced with the inconsistent explanations and/or selecting explanation method is checking whether data is correlated. If it is, then we need to establish the goal of explanation: if it is the model and its behavior we are trying to explain, then the desired explanation is "true to the model", and we should be using marginal methods (methods utilizing marginal distributions when generating explanations); if it is the data, then the explanation is "true to the data", and we should be using conditional methods (methods utilizing conditional distributions). The former is used when we want to debug, analyze, get insights of the model; latter - when we want to get insights of the data through the model, e.g. we want to get insights in biology or physics with the means of explainable Machine Learning. With conditional methods we should also be careful when interpreting explanation, because the effect of a single feature mixes effects of correlated features.

Examples of such marginal-conditional method pairs are, for example:

- PDP and M plot. PDP uses marginal distribution and shows the effect of a single feature ignoring its possible dependences with other features. M plot uses conditional distribution and shows the effect of a feature under the assumption that other features are conditioned on the given feature. There is also ALE plot that also uses conditional distribution, but it does not mix effects of correlated features as it computes local effects, but its requires a bit more difficult interpretation than PDP and M plot.
- TreeSHAP with different `feature_perturbation` parameter. It supports both marginal and conditional expectations.

If data is not correlated, marginal and conditional methods should produce the same result.

Disagreements in local explanations on tabular data

In the theoretical part we have gave an overview of the disagreement metrics from the "The Disagreement Problem" paper. They could be very useful when checking for the disagreement in local explanations produced by different explanation methods on tabular data. We can compute the average disagreement metrics over testing data to know what disagreement we can expect on

the data the model has not yet seen. If the disagreement between two methods is not high, we can choose the one that suits our task the most. Considering that there are 6 different metrics, we would be focusing on the *feature* and *sign agreement* metrics when number of features in explanation is low, because in this case it is essentially the top features we are interested and their absolute ordering. If the number of features in the explanation is high, we would be focusing more on the metrics that give importance to the relative order in explanations: *rank correlation* and *pairwise rank agreement*. Deciding what constitutes as high or low disagreement depends on the data, use case and metrics, however it should not be difficult because the metrics are pretty natural and self-explanatory.

Otherwise, if the disagreement is high, when choosing between SHAP and LIME methods, our personal preference comes to SHAP because (1) it outputs feature attributions and not only importances as does LIME, (2) each feature attribution incorporates interactions with other feature values, since the resultant attribution is, in fact, a Shapley value, and (3) LIME has a kernel width hyperparameter which can be difficult to choose since it changes the vicinity of the data point we are explaining within which LIME approximates our prediction function with linear model.

SHAP framework

We have to note that SHAP has become a versatile framework which provides a range of different explanation results such as local and global feature importances, feature dependence plots, all based on Shapley values which should minimize the inconsistencies which we would get if we used different methods for different explanations. For example, we can use PDP plot to visualize the effect of a feature onto the target variable, PFI to see the most important features and KernelSHAP to generate local explanations. The problem with this approach is that all three methods are based on different theories, and one must keep it in mind when interpreting results of each, and results might be inconsistent. Instead, SHAP could be used in all 3 scenarios, interpretation becomes easier, and neither result will have inconsistencies.

Chapter 5
Conclusion

The aim of this thesis was to give a general overview of the current state of the Explainability methods in the field of machine learning and evaluate these methods in different scenarios.

In the theoretical part we reviewed most-used model-agnostic methods for tabular data and model-specific methods for neural networks that work with images. For the methods that work with tabular data we tried to give an overview how and why these methods might be impacted by the correlation. In the theoretical part we showed that, even though correlation can be the reason for inconsistent explanations, the correlation by itself is not a problem and choosing the method for correlation is application-dependent: desired explanation could be either "true to the model" when the goal is to explain, analyze, debug the model, or "true to the data" when the goal is to get insights of the data through the model. We also gave a brief summary of the rising problem in the xAI - "The disagreement problem" - problem when different explainability methods produce explanations contradicting with each other.

In the practical part we have evaluated PDP, M Plot and ALE plot on the on the linear and non-linear models trained on artificial correlated data and showed when their results are contradicting. We have also compared SHAP and LIME methods on regression and classification datasets using the metrics described in the theoretical part which were taken from "The disagreement problem" and shown that on the simple dataset - Penguin Classification with only 6 features - results were much more consistent than on more complex regression dataset - House Price prediction with 12 features. We have also compared SHAP and LIME methods on the sparse model and seen that while SHAP could correctly identify that the model did not use some of the features, LIME could not. Then, we have compared model-agnostic KernelSHAP and model-specific TreeSHAP methods on the same datasets and shown that these two methods show much more agreement overall. Furthermore, we have tried to investigate the correlation impact on different versions of SHAP. To do that we have taken 3 models trained on 3 different datasets, and compared the mean absolute error between a given SHAP method and exact SHAP values across all dataset and features (and classes for classification data). The results were that TreeSHAP with `interventional` parameter has much lesser, almost negligible, error when compared to KernelSHAP and TreeSHAP with `tree_path_dependent` parameter. Finally, for the neural network methods we have taken MNIST dataset and given practical examples on how we can use 2 different methods - Saliency Maps and Influential Functions - to analyze and find biases in the CNN model.

In the final chapter, we are giving a list of recommendations when dealing with inconsistent outputs based on what we have reviewed in the theoretical part and our own experiments from the practical part.

Bibliography

1. RUDIN, Cynthia; WANG, Caroline; COKER, Beau. The Age of Secrecy and Unfairness in Recidivism Prediction. *Harvard Data Science Review* [https://hdsr.mitpress.mit.edu/pub/7z10o269]. 2020, vol. 2, no. 1. [Online; accessed 10-January-2024].
2. MOLNAR, Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. Available also from: https://christophm.github.io/interpretable-ml-book.
3. MOLNAR, Christoph. *Correlation can ruin interpretability*. Mindful Modeler, 2022. Available also from: https://mindfulmodeler.substack.com/p/correlation-can-ruin-interpretability. [Online, accessed 10-January-2024].
4. KRISHNA, Satyapriya; HAN, Tessa; GU, Alex; POMBRA, Javin; JABBARI, Shahin; WU, Steven; LAKKARAJU, Himabindu. *The Disagreement Problem in Explainable Machine Learning: A Practitioner's Perspective*. 2022. Available from arXiv: 2202.01602 [cs.LG].
5. SELBST, Andrew D; POWLES, Julia. Meaningful information and the right to explanation. *International Data Privacy Law*. 2017, vol. 7, no. 4, pp. 233–242. ISSN 2044-3994. Available from DOI: 10.1093/idpl/ipx022.
6. LUNDBERG, Scott; LEE, Su-In. *A Unified Approach to Interpreting Model Predictions*. 2017. Available from arXiv: 1705.07874 [cs.AI].
7. RIBEIRO, Marco Tulio; SINGH, Sameer; GUESTRIN, Carlos. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. Available from arXiv: 1602.04938 [cs.LG].
8. FRIEDMAN, Jerome H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*. 2001, vol. 29, no. 5, pp. 1189–1232. Available from DOI: 10.1214/aos/1013203451.
9. SIMONYAN, Karen; VEDALDI, Andrea; ZISSERMAN, Andrew. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. 2014. Available from arXiv: 1312.6034 [cs.CV].
10. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.
11. APLEY, Daniel W.; ZHU, Jingyu. *Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models*. 2019. Available from arXiv: 1612.08468 [stat.ME].

12. BREIMAN, Leo. *Machine Learning*. 2001, vol. 45, no. 1, pp. 5–32. ISSN 0885-6125. Available from DOI: 10.1023/a:1010933404324.

13. SHAPLEY, Lloyd S. *Notes on the N-Person Game mdash; I: Characteristic-Point Solutions of the Four-Person Game*. Santa Monica, CA: RAND Corporation, 1951. Available from DOI: 10.7249/RM0656.

14. COVERT, Ian. *Explaining machine learning models with shap and sage*. 2020. Available also from: https://iancovert.com/blog/understanding-shap-sage/. [Online; accessed 10-January-2024].

15. CONTRIBUTORS, Wikipedia. *Shapley values*. Wikimedia Foundation, 2019. Available also from: https://en.wikipedia.org/wiki/Shapley_value. [Online; accessed 10-January-2024].

16. ŠTRUMBELJ, Erik; KONONENKO, Igor. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*. 2013, vol. 41, no. 3, pp. 647–665. ISSN 0219-3116. Available from DOI: 10.1007/s10115-013-0679-x.

17. CHEN, Tianqi; GUESTRIN, Carlos. XGBoost: A Scalable Tree Boosting System. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016. Available from DOI: 10.1145/2939672.2939785.

18. LARSON, Jeff; ANGWIN, Julia; KIRCHNER, Lauren; MATTU, Surya. *How we analyzed the compas recidivism algorithm*. 2016. Available also from: https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm. [Online; accessed 10-January-2024].

19. HOFMANN, Hans. *Statlog (German Credit Data)* [UCI Machine Learning Repository]. 1994. DOI: https://doi.org/10.24432/C5NC77.

20. CHEN, Hugh; JANIZEK, Joseph D.; LUNDBERG, Scott; LEE, Su-In. *True to the Model or True to the Data?* 2020. Available from arXiv: 2006.16234 [cs.LG].

21. KOH, Pang Wei; LIANG, Percy. *Understanding Black-box Predictions via Influence Functions*. 2020. Available from arXiv: 1703.04730 [stat.ML].

22. ZEILER, Matthew D; FERGUS, Rob. *Visualizing and Understanding Convolutional Networks*. 2013. Available from arXiv: 1311.2901 [cs.CV].

23. SMILKOV, Daniel; THORAT, Nikhil; KIM, Been; VIÉGAS, Fernanda; WATTENBERG, Martin. *SmoothGrad: removing noise by adding noise*. 2017. Available from arXiv: 1706.03825 [cs.LG].

24. SELVARAJU, Ramprasaath R.; COGSWELL, Michael; DAS, Abhishek; VEDANTAM, Ramakrishna; PARIKH, Devi; BATRA, Dhruv. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*. 2019, vol. 128, no. 2, pp. 336–359. ISSN 1573-1405. Available from DOI: 10.1007/s11263-019-01228-7.

25. HORST, Allison Marie; HILL, Alison Presmanes; GORMAN, Kristen B. *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*. 2020. Available from DOI: 10.5281/zenodo.3960218.

26. KLAISE, Janis; LOOVEREN, Arnaud Van; VACANTI, Giovanni; COCA, Alexandru. Alibi Explain: Algorithms for Explaining Machine Learning Models. *Journal of Machine Learning Research*. 2021, vol. 22, no. 181, pp. 1–7. Available also from: http://jmlr.org/papers/v22/21-0017.html.

27. *TreeSHAP is not exact*. 2019. Available also from: https://github.com/christophM/interpretable-ml-book/issues/142.

28. DENG, Li. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*. 2012, vol. 29, no. 6, pp. 141–142.

29. ALBER, Maximilian; LAPUSCHKIN, Sebastian; SEEGERER, Philipp; HÄGELE, Miriam; SCHÜTT, Kristof T.; MONTAVON, Grégoire; SAMEK, Wojciech; MÜLLER, Klaus-Robert; DÄHNE, Sven; KINDERMANS, Pieter-Jan. iNNvestigate Neural Networks! *Journal of Machine Learning Research*. 2019, vol. 20, no. 93, pp. 1–8. Available also from: `http://jmlr.org/papers/v20/18-540.html`.

Contents of enclosed CD

- `src` — source code for the practical part
  - `data` — directory with the external datasets
  - `helpers` — helper packages used in the notebooks
  - `models` — saved CNN models used in the notebooks
  - `notebooks` — notebooks
- `thesis`
  - `src` — source files of the thesis
  - `thesis.pdf` — compiled thesis