Effector: A Python package for regional explanations

Vasilis Gkolemis[*,†] GKOLEMIS@HUA.GR, VGKOLEMIS@ATHENARC.GR
Julia Herbinger[¶] JULIA.HERBINGER@STAT.UNI-MUENCHEN.DE
Christos Diou[*] DIOU@HUA.GR
Theodore Dalamagas[†] DALAMAG@ATHENARC.GR
Eirini Ntoutsi[§] EIRINI.NTOUTSI@UNIBW.DE
Bernd Bischl[¶] BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Giuseppe Casalicchio[¶] GIUSEPPE.CASALICCHIO@LMU.DE
[*] *Harokopio University of Athens,* [†] *ATHENA RC,* [§] *University of the Bundeswehr Munich*
[¶] *Munich Center for Machine Learning (MCML), Department of Statistics, LMU Munich*

Abstract

Global feature effect methods explain a model outputting one plot per feature. The plot shows the average effect of the feature on the output, like the effect of age on the annual income. However, average effects may be misleading when derived from local effects that are heterogeneous, i.e., they significantly deviate from the average. To decrease the heterogeneity, regional effects provide multiple plots per feature, each representing the average effect within a specific subspace. For interpretability, subspaces are defined as hyperrectangles defined by a chain of logical rules, like age's effect on annual income separately for males and females and different levels of professional experience. We introduce `Effector`, a Python library dedicated to regional feature effects. `Effector` implements well-established global effect methods, assesses the heterogeneity of each method and, based on that, provides regional effects. `Effector` automatically detects subspaces where regional effects have reduced heterogeneity. All global and regional effect methods share a common API, facilitating comparisons between them. Moreover, the library's interface is extensible so new methods can be easily added and benchmarked. The library has been thoroughly tested, ships with many tutorials (https://xai-effector.github.io/) and is available under an open-source license at PyPi https://pypi.org/project/effector/ and Github https://github.com/givasile/effector.

Keywords: Explainability, interpretability, global explanations, regional explanations.

1 Introduction

The increasing adoption of machine learning (ML) in high-stakes domains like healthcare and finance has raised the demand for explainable AI (XAI) techniques (Freiesleben et al.; Ribeiro et al., 2016). Global feature effect methods explain a black-box model through a set of plots, where each plot is the effect of a feature on the output, as in Figure 1a.

Global effects may be misleading when the black-box model $f(\cdot)$ exhibits interactions between features. An interaction between two features, x_s and x_k, exists when the difference in the output $f(\mathbf{x})$ as a result of changing the value of x_s *depends* on the value of x_k (Friedman and Popescu, 2008). Global effects are often computed as averages over local effects. When feature interactions are present, local effects become heterogeneous, i.e., they significantly deviate from the average (global) effect. In these cases, the global effect may

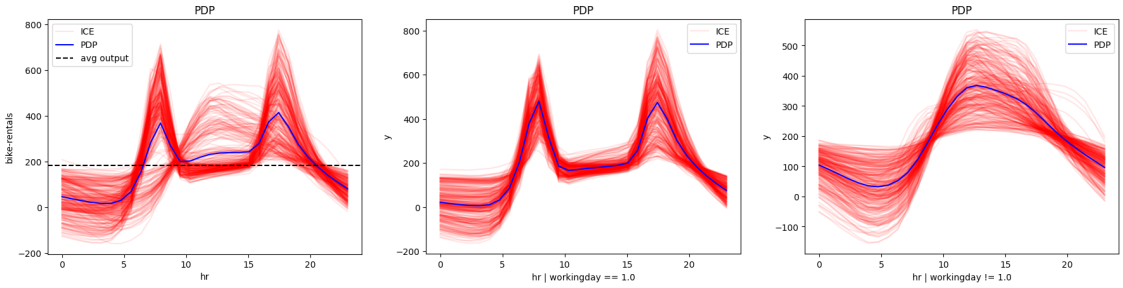
be misleading, a phenomenon known as *aggregation bias* (Mehrabi et al., 2021; Herbinger et al., 2022).

Regional (Herbinger et al., 2023, 2022; Molnar et al., 2023; Britton, 2019; Hu et al., 2020; Scholbeck et al., 2022) or cohort explanations (Sokol and Flach, 2020), partition the input space into subspaces and compute a regional explanation within each. The partitioning aims at subspaces with homogeneous local effects, i.e., with reduced feature interactions, yielding regional effects with minimized aggregation bias (Herbinger et al., 2022). By combining these regional explanations, users can interpret the model's behavior across the entire input space. Several libraries focus on XAI, but none targets on regional effect methods (Table 1). Therefore, we present `Effector`, a Python library dedicated to regional explainability methods, which:

- implements well-established global and regional effect methods, accompanied by an intuitive way to visualize the heterogeneity of each plot.
- follows a consistent and modular software design. Existing methods share a common API and novel methods can be easily added and compared to existing ones.
- demonstrates through tutorials the use of regional effects in real and synthetic datasets. Synthetic datasets we evaluate the implemented methods with ground truth.

2 `Effector` - A python package for global and regional feature effects

To showcase the effectiveness of `Effector` we use the Bike-Sharing dataset (Fanaee-T and Gama, 2014). The results are plot in Figure 1 and explained hereafter.



(a) Global effect (b) Regional on weekdays (c) Regional on weekends

Figure 1: `RegionalPDP` applied to the Bike Sharing dataset; Left: Global effect of feature `hour` on the `bike-rentals`. Middle: Regional effect of feature `hour` on weekdays. Right: Regional effect of feature `hour` on weekends.

Regional Effects. Global effect methods are vulnerable to aggregation bias. Figure 1a shows that the global effect (blue curves) of feature `hour` on the `bike-rentals` in the Bike-Sharing dataset (Fanaee-T and Gama, 2014) has two sharp peaks at about 7:30 and 17:00 which indicate people's commute patterns to and from work. The ICE plots (red curves), however, indicate heterogeneity; there are many instances that deviate from this explanation. Regional effect plots automatically detect two subspaces; one for weekdays

(Figure 1b) with a peak at about 7:30 and 17:00 and one for weekends (Figure 1c) with a peak at about 13:00-16:00, with decreased heterogeneity compared to the global effect. The example highlights that regional effects can uncover different explanations that are hiding behind the global (average) effect.

Methods. Figure 1 is obtained using `PDP` and `RegionalPDP`, but similar findings are obtained with any other method, as shown in Appendix D. `Effector` implements the following global effect methods alongside their regional counterparts; Partial Dependence Plots (PDP) (Friedman, 2001), derivative-PDP (Goldstein et al., 2014), Accumulated Local Effects (ALE) (Apley and Zhu, 2020), Robust and Heterogeneity-aware ALE (RHALE) (Gkolemis et al., 2023a) and SHAP Dependence Plots (SHAP-DP) (Lundberg and Lee, 2017). Formal descriptions of these methods are provided in Appendices A and B. Comparative analyses using synthetic examples with ground truth effects are detailed in Appendix C, while their application to real datasets is presented in Appendix D. Finally, instructions on integrating Effector with common machine learning libraries are outlined in Appendix E.

Usage. `Effector` adheres to the progressive disclosure of complexity; simplifies the initial step and demands incremental learning steps for increasingly sophisticated use cases. Global or regional plots require a single-line command (lines 10 and 11 of Listing 1). In three lines of code, the user can customize the bin-splitting of RHALE (line 15) or change the heterogeneity threshold for an accepted split (line 20). All methods require two arguments: the dataset `X` and the predictive function `f`. For RHALE and d-PDP, the user can accelerate the computation, providing a function `f_der` that computes the derivatives with respect to the input. Otherwise, finite differences (Scholbeck et al., 2022) will be used. `Effector` is compatible with any underlying ML library; it only requires wrapping the trained model into a Python `Callable`, as shown in Appendix E.

```
1  from effector import RHALE, RegionalRHALE
2  from effector.binning_methods import DynamicProgramming
3
4  # Input
5  # X = .., the dataset (np.array of shape=(N,D))
6  # f = .., predictive function (python callable)
7  # f_der = .., predictive function's jacobian (python callable)
8
9  # 1-step global and regional effect
10 RHALE(X, f, f_der).plot(feature=1)
11 RegionalRHALE(x, f, f_der).plot(feature=1, node_idx=0)
12
13 # 3-step global effect
14 rhale = RHALE(X, f, f_der)
15 rhale.fit(binning_method=DynamicProgramming(max_nof_bins=100))
16 rhale.plot(feature=1)
17
18 # 3-step regional effect
19 regional_rhale = RegionalRHALE(X, f, f_der)
20 regional_rhale.fit(heter_pcg_drop_thres=0.1)
21 rhale.plot(feature=1, node_idx=0)
```

Listing 1: Listing showing how to use `Effector`.

Design The domain of regional effects is rapidly evolving, so `Effector` is designed to maximize extensibility. The library is structured around a set of abstractions that facilitate the development of novel methods. For example, a user can easily experiment with a novel ALE-based approach by subclassing the `ALEBase` class and redefining the `.fit()` and `.plot()` methods. Or they can define a new RHALE-based heterogeneity index by subclassing the `RegionalRHALE` class.

Comparison to existing frameworks Global effect methods, such as PDP and ALE, are integrated into various explainability toolkits (Pedregosa et al., 2011; Klaise et al., 2021; Baniecki et al., 2021). However, none of these toolkits implements all methods, and notably, none includes their regional counterparts, as shown in Table 1. Only DALEX offers a feature for grouping ICE plots using K-means, akin to subspace identification. As we can see, only DALEX provides functionality for grouping ICE plots using K-means, something similar to subspace detection. However, K-means clustering results in less interpretable subspaces and, notably, DALEX lacks support for the other regional effect methods.

Table 1: Comparison between current feature effect libraries and `Effector`

	(d-)PDP		(RH)ALE		SHAP-DP	
	Global	Regional	Global	Regional	Global	Regional
`InterpretML`	✓	✗	✗	✗	✗	✗
`DALEX`	✓	*	✓	✗	✗	✗
`ALIBI`	✓	✗	✓	✗	✗	✗
`PyALE`	✗	✗	✓	✗	✗	✗
`ALEPython`	✗	✗	✓	✗	✗	✗
`PDPbox`	✓	✗	✗	✗	✗	✗
`Effector`	✓	✓	✓	✓	✓	✓

Broader Impact The principle of progressive disclosure of complexity ensures `Effector`'s applicability in both research and industrial contexts. Users can easily get regional explanations for black-box models trained on tabular data in a single-line. Regional methods that exploit automatic differentiation, like RHALE, will be particularly useful for Deep Learning models on tabular data (Agarwal et al., 2021). Researchers can leverage `Effector` to develop novel methods and compare them with the existing approaches. The library will keep being updated to incorporate emerging regional effect methods.

3 Conclusion and Future Work

We introduced `Effector`, a Python library dedicated to regional explainability methods. The library is accessible at PyPi. `Effector` ships with comprehensive documentation and tutorials to familiarize the community with the realm of regional explainability methods, https://xai-effector.github.io/ Immediate plans involve accommodating multiple features of interest in the regional effect methods, as in GADGET (Herbinger et al., 2023) and to encompass other types of regional explainability methods, such as methods based on marginal effects (Scholbeck et al., 2022).

Appendix A. Global Effect Methods at `Effector`

This section provides a brief mathematical overview of the global effect methods that are implemented at `Effector`. For an in-depth understanding of the methods, we refer the reader to the original papers (Friedman and Popescu, 2008; Goldstein et al., 2014; Apley and Zhu, 2020; Gkolemis et al., 2023a; Lundberg and Lee, 2017; Herbinger et al., 2023, 2022).

Each global effect method consists of a *definition* $f_s^{\mathtt{m}}(x_s)$ and an *approximation* $\hat{f}_s^{\mathtt{m}}(x_s)$, where $\mathtt{m}$ is the method's name and x_s is the feature of interest. The *definition* derives the effect of the s-th feature on the output based on the data-generating distribution $p(\mathbf{x})$ and the analytical form of the predictive function f. The approximation, on the other hand, estimates the effect based on the dataset instances $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and the predictive function f treated as an input-output black-box. In real-world scenarios, only the approximation is applicable, as the data-generating distribution $p(\mathbf{x})$ and the analytical form of f are unknown.

Furthermore, each method provides a way to quantify the heterogeneity of the local effects, i.e., the deviation of the local effects from the average effect. Since there is variation in how each method defines and visualizes the heterogeneity, it is difficult to provide unified mathematical notation. However, for all methods, we define a heterogeneity index $\hat{H}_s^{\mathtt{m}}$, which quantifies the level of interactions between the s-th and the rest of the features and is computed using the dataset instances. This index will be later used to find the regional effects. If the method additionally provides a formal definition of the heterogeneity index using the data-generating distribution, we denote it as $H_s^{\mathtt{m}}$. Finally, if a method also provides a way to quantify the heterogeneity at each point along the s-th axis, apart from a global index, we denote it as $H_s^{\mathtt{m}}(x_s)$ and $\hat{H}_s^{\mathtt{m}}(x_s)$, respectively.

For all methods, `Effector` implements the approximation of the global effect and the heterogeneity index for each method, which we summarize at Table 2.

Notation. Let $\mathcal{X} \in \mathbb{R}^d$ be the d-dimensional feature space, $\mathcal{Y}$ the target space and $f(\cdot) : \mathcal{X} \to \mathcal{Y}$ the black-box function. We use index $s \in \{1, \dots, d\}$ for the feature of interest and $/s = \{1, \dots, d\} - s$ for the rest. For convenience, we use $(x_s, \mathbf{x_c})$ to denote the input vector $(x_1, \cdots, x_s, \cdots, x_D)$, $(X_s, \mathbf{X_c})$ instead of $(X_1, \cdots, X_s, \cdots, X_D)$ for random variables and $\mathcal{X}_s, \mathcal{X}_c$ for the feature space and its complement, respectively. The training set $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^N$ is sampled i.i.d. from the distribution $\mathbb{P}_{X,Y}$.

A.1 ALE and RHALE

Accumulated Local Effects (ALE) was introduced by (Apley and Zhu, 2020). Subsequently, (Gkolemis et al., 2023a) introduced Robust and Heterogeneity-aware ALE (RHALE) which extends ALE's *definition* with a component that quantifies the heterogeneity. Furthermore, RHALE introduced a novel *approximation* of both the global effect and the heterogeneity using an automated bin-splitting. ALE and RHALE can handle datasets with correlated features well, as we will illustrate below. In contrast, (d-)PDP and SHAP-DP have limitations in these cases as we will see in Sections A.2 and A.3.

Table 2: Global effect methods implemented by `Effector`

Method	Equation	Formula
$\hat{f}^{\texttt{RHALE}}(x_s)$	Eq. (4)	$\sum_{k=1}^{kx_{x_s}} \frac{z_k - z_{k-1}}{\lvert\mathcal{S}_k\rvert} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \frac{\partial f}{\partial x_s}(x_s^i, \mathbf{x}_c^i)$
$\hat{H}_s^{\texttt{RHALE}}$	Eq. (6)	$\sum_{k=1}^{K_s} \frac{z_k - z_{k-1}}{\lvert\mathcal{S}_k\rvert} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[\frac{\partial f}{\partial x_s}(x_s^i, \mathbf{x}_c^i) - \hat{\mu}_k^{\texttt{RHALE}}\right]^2$
$\hat{f}^{\texttt{ALE}}(x_s)$	Eq. (3)	$\sum_{k=1}^{k_{x_s}} \frac{1}{\lvert\mathcal{S}_k\rvert} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[f(z_k, \mathbf{x}_c^i) - f(z_{k-1}, \mathbf{x}_c^i)\right]$
$\hat{H}_s^{\texttt{ALE}}$	Eq. (5)	$\sum_{k=1}^{K} \frac{1}{\lvert\mathcal{S}_k\rvert} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[(f(z_k, \mathbf{x}_c^i) - f(z_{k-1}, \mathbf{x}_c^i)) - \hat{\mu}_k^{\texttt{ALE}}\right]^2$
$\hat{f}^{\texttt{PDP}}(x_s)$	Eq. (7)	$\frac{1}{N} \sum_{i=1}^{N} f(x_s, \mathbf{x}_c^i)$
$\hat{H}_s^{\texttt{PDP}}$	Eq. (8)	$\frac{1}{T} \sum_{t=1}^{T} \frac{1}{N} \sum_{i=1}^{N} \left[\hat{f}_{i,\text{centered}}^{\texttt{ICE}}(x_{s,t}) - \hat{f}_{\text{centered}}^{\texttt{PDP}}(x_{s,t})\right]^2$
$\hat{f}^{\texttt{d-PDP}}(x_s)$	Eq. (9)	$\frac{1}{N} \sum_{i=1}^{N} f(x_s, \mathbf{x}_c^i)$
$\hat{H}_s^{\texttt{d-PDP}}$	Eq. (10)	$\frac{1}{T} \sum_{t=1}^{T} \frac{1}{N} \sum_{i=1}^{N} \left[\hat{f}_{i,\text{centered}}^{\texttt{ICE}}(x_{s,t}) - \hat{f}_{\text{centered}}^{\texttt{PDP}}(x_{s,t})\right]^2$
$\hat{f}^{\texttt{SHAP-DP}}(x_s)$	Eq. (13)	$\kappa(x_s)$, $\quad \kappa(x_s)$ is a univariate spline fit to $\{(x_s^i, \hat{\phi}_s^i)\}_{i=1}^N$
$\hat{H}_s^{\texttt{SHAP-DP}}$	Eq. (14)	$\frac{1}{N} \sum_{i=1}^{N} \left[\hat{\phi}_s^i - f^{\texttt{SHAP-DP}}(x_s^i)\right]^2$

ALE defines the global effect at x_s as an accumulation of the derivative effects. The derivative effect $\mu(x_s)$ is the expected change on the output over the conditional distribution $\mathbf{x_c}|x_s$, i.e., $\mu(x_s) = \mathbb{E}_{\mathbf{x_c}|x_s}\left[\frac{\partial f}{\partial x_s}(x_s, \mathbf{x_c})\right]$. ALE's *definition* is:

$$f^{\texttt{ALE}}(x_s) = \int_{z=0}^{x_s} \mathbb{E}_{\mathbf{x_c}|x_s=z}\left[\frac{\partial f}{\partial x_s}(z, \mathbf{x_c})\right] \partial z \tag{1}$$

and is visualized with a $1D$ plot as shown in Figure 2 (Left). RHALE *definition* is identical to ALE, i.e., $f^{\texttt{RHALE}}(x_s) := f^{\texttt{ALE}}(x_s)$. Additionally, RHALE quantifies the heterogeneity $H_s^{\texttt{RHALE}}(x_s)$ as the standard deviation of the derivative effects:

$$H_s^{\texttt{RHALE}}(x_s) = \sigma_{\mathbf{x_c}|x_s}\left[\frac{\partial f}{\partial x_s}(x_s, \mathbf{x_c})\right] \tag{2}$$

The heterogeneity is visualized as a shaded area around the derivative effect. Therefore, RHALE's *definition* proposes two separate plots; one for the average effect which is identical to ALE (Figure 2 (Left)) and another for the derivative effect along with its associated heterogeneity, i.e., $\mu(x_s) \pm H(x_s)$ (Figure 2 (Right)). The two plots are arranged one under

the other. The heterogeneity index is the aggregated heterogeneity over the s-th axis, i.e., $H_s^{\texttt{RHALE}} = \int_{z=0}^{x_s} H^{\texttt{RHALE}}(z)\partial z$. ALE's *approximation* is:

$$\hat{f}^{\texttt{ALE}}(x_s) = \sum_{k=1}^{k_{x_s}} \underbrace{\frac{1}{|\mathcal{S}_k|} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[f(z_k, \mathbf{x}_c^i) - f(z_{k-1}, \mathbf{x}_c^i) \right]}_{\hat{\mu}_k^{\texttt{ALE}}} \tag{3}$$

where k_{x_s} the index of the bin such that $z_{k_{x-1}} \leq x_s < z_{k_x}$, $\mathcal{S}_k$ is the set of the instances lying at the k-th bin, i.e., $\mathcal{S}_k = \{\mathbf{x}^i : z_{k-1} \neq x_s^{(i)} < z_k\}$ and $\Delta x = \frac{x_{s,max} - x_{s,min}}{K}$. In ALE, the user needs to specify the number K of equally-sized bins that are used to split the feature axis. RHALE resolves that by approximating the global effect as:

$$\hat{f}^{\texttt{RHALE}}(x_s) = \sum_{k=1}^{k_{x_s}} \underbrace{\frac{z_k - z_{k-1}}{|\mathcal{S}_k|} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \frac{\partial f}{\partial x_s}(x_s^i, \mathbf{x}_c^i)}_{\hat{\mu}_k^{\texttt{RHALE}}} \tag{4}$$

RHALE's *approximation* differentiates from ALE *approximation* in two aspects. First, it computes the derivative effects using automatic differentiation instead of finite differences at the bin limits. This estimates the derivative effects more accurately and faster (Gkolemis et al., 2022, 2023b). Second, it automatically partitions the s-th axis into a sequence of K_s variable-size intervals, i.e., $\{\mathcal{Z}_k\}_{k=1}^{K_s}$. Each interval covers a range from z_{k-1} to z_k, and defines the set $\mathcal{S}_k$ with the instances that lie inside, i.e., $\mathcal{S}_k = \{x^{(i)} : z_{k-1} \leq x_s^{(i)} < z_k\}$. The automatic bin-splitting is performed by solving an optimization problem that minimizes the heterogeneity of the derivative effect within each bin (Gkolemis et al., 2023a). The *approximations* of the heterogeneity index is given by:

$$\hat{H}_s^{\texttt{ALE}} = \sum_{k=1}^{K} \frac{1}{|\mathcal{S}_k|} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[(f(z_k, \mathbf{x}_c^i) - f(z_{k-1}, \mathbf{x}_c^i)) - \hat{\mu}_k^{\texttt{ALE}} \right]^2 \tag{5}$$

$$\hat{H}_s^{\texttt{RHALE}} = \sum_{k=1}^{K_s} \frac{z_k - z_{k-1}}{|\mathcal{S}_k|} \sum_{i:\mathbf{x}^i \in \mathcal{S}_k} \left[\frac{\partial f}{\partial x_s}(x_s^i, \mathbf{x}_c^i) - \hat{\mu}_k^{\texttt{RHALE}} \right]^2 \tag{6}$$

A.2 PDP, d-PDP, ICE, d-ICE

PDP (Friedman and Popescu, 2008) is probably the most widely used global effect method. As a way to visualize the heterogeneity, (Goldstein et al., 2014) introduced ICE plots which generate one plot per dataset instance and visualize them on top of the PDP, as shown in Figure 5(Left). Moreover, (Goldstein et al., 2014) presented d-PDP and d-ICE, the derivative counterparts of PDP and ICE, respectively. The d-PDP and d-ICE are used to visualize the derivative effect of a feature, as shown in Figure 5(Right). Both PDP and d-PDP, however, may produce misleading explanations when applied to correlated features, as we discuss below.

PDP computes the average effect of a feature by averaging the predictions over the entire dataset while holding the feature of interest constant. The PDP *definition* is given by $f^{\texttt{PDP}}(x_s) = \mathbb{E}_{\mathbf{x_c}}\left[f(x_s, \mathbf{x_c})\right]$ and the *approximation* by:

$$\hat{f}^{\texttt{PDP}}(x_s) = \frac{1}{N}\sum_{i=1}^{N} f(x_s, \mathbf{x}_c^i) \tag{7}$$

The ICE plots are used on top of the PDP plot to visualize the heterogeneity of the instance-level effects. One ICE plot is defined per dataset instance, holding all features constant except of the feature of interest. The ICE plot of the i-th instance is given by $\hat{f}_i^{\texttt{ICE}}(x_s) = f(x_s, \mathbf{x_c}^{(i)})$.

The heterogeneity index is the average squared difference between the centered-ICE, $\hat{f}_{i,\text{centered}}^{\texttt{ICE}}(x_s) = \hat{f}_i^{\texttt{ICE}}(x_s) - \mathbb{E}_{x_s}\left[\hat{f}_i^{\texttt{ICE}}(x_s)\right]$, and the centered-PDP plots, $\hat{f}_{\text{centered}}^{\texttt{PDP}}(x_s) = \hat{f}^{\texttt{PDP}}(x_s) - \mathbb{E}_{x_s}\left[\hat{f}^{\texttt{PDP}}(x_s)\right]$. The reason for centering the plots is analyzed in (Herbinger et al., 2023). The heterogeneity index is computed by splitting the s-th axis into a sequence of T points, i.e., $\{x_{s,t}\}_{i=1}^{T}$, where $x_{s,t} = x_{s,min} + t \cdot \Delta x$ and $\Delta x = \frac{x_{s,max} - x_{s,min}}{T}$:

$$\hat{H}_s^{\texttt{PDP}} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{i=1}^{N}\left[\hat{f}_{i,\text{centered}}^{\texttt{ICE}}(x_{s,t}) - \hat{f}_{\text{centered}}^{\texttt{PDP}}(x_{s,t})\right]^2 \tag{8}$$

The d-PDP and d-ICE are the derivative counterparts to PDP and ICE. They calculate the derivative effect of a feature by averaging predictions across the entire dataset while keeping the feature of interest constant. The d-PDP is defined as $f^{\texttt{d-PDP}}(x_s) = \mathbb{E}_{\mathbf{x_c}}\left[\frac{\partial f}{\partial x_s}(x_s, \mathbf{x_c})\right]$ and approximated by:

$$\hat{f}^{\texttt{d-PDP}}(x_s) = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial f}{\partial x_s}(x_s, \mathbf{x_c}^{(i)}) \tag{9}$$

The d-ICE plots $\hat{f}^{\texttt{d-ICE}}(x_s) = \frac{\partial f}{\partial x_s}(x_s, \mathbf{x_c}^{(i)})$ are used on top of the d-PDP plot to visualize the heterogeneity of the instance-level derivative effects. The heterogeneity index of the d-PDP is given by:

$$\hat{H}_s^{\texttt{d-PDP}} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{i=1}^{N}\left[\hat{f}_i^{\texttt{d-ICE}}(x_{s,t}) - \hat{f}^{\texttt{d-PDP}}(x_{s,t})\right]^2 \tag{10}$$

Note that the heterogeneity index for d-PDP does not require the centered plots, as the derivative effects are centered by definition. For a detailed explanation about that, we refer the reader to (Herbinger et al., 2023).

A.3 SHAP Dependence Plot

SHAP Dependence Plot (SHAP-DP) is a recent method introduced by (Lundberg and Lee, 2017). SHAP-DP builds upon SHAP values, a local explainability method used to measure the contribution of the s-th feature of the i-th instance on the output, denoted as ϕ_s^i:

$$\phi_s^i = \frac{|Q|!(p-|Q|-1)!}{p!} \sum_{Q \subseteq \{1,\dots,p\} \setminus \{s\}} (v(Q \cup s) - v(Q)) \tag{11}$$

At the core of Eq. (11) is the *value* of a coalition of Q features, given by $v(Q)$. The value of a coalition Q is the expected value of the output of the model if only the values of the features in Q are known and the rest are considered unknown, i.e., $v(Q) = \mathbb{E}_{\mathbf{X}_{/Q}}\left[f(\mathbf{x}_Q, \mathbf{X}_{/Q})\right]$. The difference $v(Q \cup \{s\}) - v(Q)$ is the contribution of the s-th feature to the coalition Q. The SHAP value ϕ_s^i is the weighted average of the contributions over all possible coalitions Q, with weight $\frac{|Q|!(p-|Q|-1)!}{p!}$. The SHAP-DP is a scatter plot with the feature values on the x-axis and the SHAP values on the y-axis and i.e., $\{(x_s^i, \phi_s^i)\}_{i=1}^N$, as shown in Figure 7. For computing an average effect of the s-th feature, (Herbinger et al., 2023) fit a univariate spline to the dataset $\{(x_s^i, \phi_s^i)\}_{i=1}^N$, so the SHAP-DP definition is:

$$f^{\texttt{SHAP-DP}}(x_s) = \kappa(x_s), \quad \kappa(x_s) \text{ is a univariate spline fit to } \{(x_s^i, \phi_s^i)\}_{i=1}^N \tag{12}$$

The number of coalitions grows exponentially with the number of features, therefore, the SHAP values are approximated using a Monte Carlo method. We denote the approximation of the SHAP values as $\hat{\phi}_s^i$. For our approximation, we use the implementation provided by the `shap` library. The `shap` library uses the exact SHAP values if the number of features is less than 10, otherwise, it uses the permutation approximation, which is a Monte Carlo sampling of coalitions. The SHAP-DP *approximation* is given by:

$$\hat{f}^{\texttt{SHAP-DP}}(x_s) = \kappa(x_s), \quad \kappa(x_s) \text{ is a univariate spline fit to } \{(x_s^i, \hat{\phi}_s^i)\}_{i=1}^N \tag{13}$$

The heterogeneity index is the mean squared difference between the SHAP values and the SHAP-DP fit:

$$\hat{H}_s^{\texttt{SHAP-DP}} = \frac{1}{N} \sum_{i=1}^{N} \left[\hat{\phi}_s^i - f^{SHAP-DP}(x_s^i)\right]^2 \tag{14}$$

Appendix B. Regional Effect Methods

This section provides a brief overview of the regional effect methods of `Effector`. For a more thorough analysis, we refer the reader to (Herbinger et al., 2023). Regional effect methods yield for each individual feature s, a set of T_s non-overlapping regions $\{\mathcal{R}_{st}\}_{t=1}^{T_s}$ where $\mathcal{R}_{st} \in \mathcal{X}_c$. The number of non-overlapping regions T_s can be different for each feature, the regions $\{\mathcal{R}_{st}\}_{t=1}^{T_s}$ are disjoint and their union covers the entire space, i.e., $\cup_{t=1}^{T_s} \mathcal{R}_{st} = \mathcal{X}_c$. Each region $\mathcal{R}_{st}$ is associated with a regional effect $\hat{f}_{st}$ and a heterogeneity $\hat{H}_{st}$, which is computed using the dataset $\mathcal{D}_{st}$ that contains the instances of $\mathcal{D}$ that fall within $\mathcal{R}_{st}$, i.e., $\mathcal{D}_{st} = \{(\mathbf{x}^i, y^i) : \mathbf{x}_c^i \in \mathcal{R}_{st}\}$.

The objective is to split $\mathcal{X}_c$ into regions inside of which the heterogeneity is minimized:

$$\begin{aligned} \underset{\{\mathcal{R}_{st}\}_{t=1}^{T_s}}{\text{minimize}} \quad & \mathcal{L}_s = \sum_{t=1}^{T_s} \frac{|\mathcal{D}_{st}|}{|\mathcal{D}|} H_{st}^{\mathtt{m}} \\ \text{subject to} \quad & \bigcup_{t=1}^{T} \mathcal{R}_{st} = \mathcal{X}_c \\ & \mathcal{R}_{st} \cap \mathcal{R}_{s\tau} = \emptyset, \quad \forall t \neq \tau \end{aligned} \tag{15}$$

$H_{st}^{\mathtt{m}}$ is the heterogeneity of the s-th feature at region t, i.e., the heterogeneity computed using the dataset $\mathcal{D}_{st}$. For convenience, we denote as $H_{s0}^{\mathtt{m}}$ the heterogeneity of the s-th feature over the entire space $\mathcal{X}_c$, i.e., $H_{s0}^{\mathtt{m}} = H_s^{\mathtt{m}}$.The formula of $H_{st}^{\mathtt{m}}$ depends on the regional effect method. Table 2 summarizes the regional effect methods implemented by `Effector`.

`Effector` solves the optimization problem (15) using Algorithm 1. The algorithm starts with the entire space $\mathcal{X}_c$ and recursively splits it into two subspaces. To describe the process, imagine the following example. For feature $s = 2$, the algorithm starts with $\{x_1, x_3\}$ as candidate split-features for the first level of the tree. For each candidate split-feature, the algorithm determines the candidate split positions.

Since x_1 is a continuous feature, the candidate splits positions p are a linearly spaced grid of P points within the range of the feature, for example, $[-1, 1]$, where P is a hyperparameter of the algorithm. Each position splits into two subspaces, i.e., $\mathcal{R}_{s1} = \{(x_1, x_3) : x_1 \leq p\}$ and $\mathcal{R}_{s2} = \{(x_1, x_3) : x_1 > p\}$ and two datasets $\mathcal{D}_{s1} = \{(\mathbf{x}^i, y^i) : x_1^i \leq p\}$ and $\mathcal{D}_{s2} = \{(\mathbf{x}^i, y^i) : x_1^i > p\}$. Each split relates to a heterogeneity index which is the weighted sum of the heterogeneity of the respective subspaces, i.e., $\sum_{t=1}^{2} \frac{|\mathcal{D}_{st}|}{|\mathcal{D}|} H_{st}^{\mathtt{m}}$.

Since x_3 is a categorical feature, the candidate splits positions p are the unique values of the feature, for example, $\{1, 2, 3\}$. Each position splits into two subspaces, i.e., $\mathcal{R}_{s1} = \{(x_1, x_3) : x_3 = p\}$ and $\mathcal{R}_{s2} = \{(x_1, x_3) : x_3 \neq p\}$ and two datasets $\mathcal{D}_{s1} = \{(\mathbf{x}^i, y^i) : x_3^i = p\}$ and $\mathcal{D}_{s2} = \{(\mathbf{x}^i, y^i) : x_3^i \neq p\}$. Each split relates to a heterogeneity index which is the weighted sum of the heterogeneity of the respective subspaces, i.e., $\sum_{t=1}^{2} \frac{|\mathcal{D}_{st}|}{|\mathcal{D}|} H_{st}^{\mathtt{m}}$.

The algorithm then selects the split that minimizes the heterogeneity index, which is assigned as the split of the first level of the tree, i.e. H_s^1. Then it continues to the next level of the tree, where it splits the subspaces of the previous level. Given the previous example, the algorithm will split the subspaces $\mathcal{R}_{s1}$ and $\mathcal{R}_{s2}$ of the first level of the tree into two subspaces each. If the weighted sum of the heterogeneity of the four subspaces reduces the heterogeneity of the previous level over the threshold ϵ (Line 8 of Algorithm 1), the algorithm will continue to the next level of the tree. If the algorithm reaches the maximum depth of the tree, it stops and returns the subspaces $\{\mathcal{R}_{st}\}_{t=1}^{2^L}$. The maximum tree level L is a hyperparameter of the algorithm, which is proposed to be set to $L = 3$ levels, i.e., a maximum of $T = 2^L = 8$ subspaces per feature.

Algorithm 1 is a general algorithm that can be used for any regional effect method; the only thing that changes between the methods is the heterogeneity index $H_{st}^{\mathtt{m}}$.

Appendix C. Synthetic Examples

In this section, we demonstrate the utilization of `Effector` through synthetic examples. The synthetic examples, where both the data-generating distribution $p(\mathbf{x})$ and the predictive

Algorithm 1: Detectsubspaces

Input : Heterogeneity function H_s, Maximum depth L

Output: subspaces $\{\mathcal{R}_{st}\}_{t=1}^{T_s}$, where $T_s \in \{0, 2, \dots, 2^L\}$

H_{s0} ; // Compute the level of interactions before any split

$D = \{(\mathbf{x}^i, y^i)\}_{i=1}^N$; // Initial dataset

$T_s = 0$; // Initialize the number of splits for feature s

for $l = 1$ *to* L **do**

  if $H_s^{l-1} = 0$ **then**

    break ; // Stop if the heterogeneity is zero

  end

  /* Iterate over all features $\mathbf{x_c}$ and candidate split positions p */

  /* Find the optimal split with heterogeneity $H_s^l = \sum_{t=1}^{2^l} \frac{|\mathcal{D}_{st}|}{|\mathcal{D}|} H_{st}$ */

  /* Define the subspaces $\{\mathcal{R}_{st}\}_{t=1}^{2^l}$ and the datasets $\{\mathcal{D}_{st}\}_{t=1}^{2^l}$ */

  if $\frac{H_s^l}{H_s^{l-1}} < \epsilon$ **then**

    break ; // Stop, if heterogeneity drop is small ($< \epsilon$)

  end

  $T_s = 2^l$; // Update the number of splits for feature s

end

return $\{\mathcal{R}_{st} | s \in \{1, \dots, D\}, t \in \{1, \dots, T_s\}\}$

function f are known, are suitable to understand the strengths and limitations of each global and regional effect method.

Synthetic examples offer many advantages. First, they enable the computation of the global and regional effects of each method based on their *definition*. Consequently, the strengths and weaknesses are clear, unaffected by potential misconceptions introduced by approximation errors. Second, it is feasible to compare different *approximations* against a known ground truth. For example, we can objectively compare the ALE *approximation* versus the RHALE *approximation* using the ALE *definition* as the ground truth. Finally, comparing with a known-ground truth also serves as a sanity check for the implementations; if the *approximation* is close to the *definition*, then the implementation is correct.

C.1 Global Effect: Comparative study of `Effector`'s methods

This example shows how to use `Effector` for generating global effect plots and assessing their heterogeneity. `Effector`'s plots are derived from the *approximation* formula of each global effect method. All plots are compared with their ground-truth, which is derived from the *definition* of each method after applying analytical derivations.

C.1.1 Problem set-up

The data generating distribution is defined as follows: $X_1 \sim p(x_1) = \frac{5}{6}\mathcal{U}(x_1; -0.5, 0) + \frac{1}{6}\mathcal{U}(x_1; 0, 0.5)$, $X_2 \sim p(x_2) = \mathcal{N}(x_2; \mu = 0, \sigma = 2)$ and $x_3 = x_1 + \epsilon$, where ϵ is sampled

from $\mathcal{N}(0, \sigma = 0.1)$. Essentially, this indicates that X_1 lies within the range $[-0.5, 0.5]$ with the $\frac{5}{6}$ of the samples falling in the first half $[-0.5, 0]$. X_2 is independent and normally distributed around around zero ($\mu_2 = 0$, $\sigma_2 = 2$), while X_3 is highly correlated with X_1. The 'black-box' function is $f(x) = \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) + x_1 x_2 + x_2$.

C.1.2 ALE and RHALE

Definition on the synthetic example. In the synthetic example, the ALE and RHALE average effect for feature x_1 is given by:

$$
\begin{aligned}
f^{\texttt{ALE}}(x_1) = f^{\texttt{RHALE}}(x_1) &= \int_{z=0}^{x_s} \mathbb{E}_{x_2,x_3|x_1=z}\left[\frac{\partial f}{\partial x_1}(z, x_2, x_3)\right] \partial z && (16)\\
&= \int_{z=0}^{x_s} \mathbb{E}_{x_2}\mathbb{E}_{x_3|z}\left[2\pi\cos(2\pi z)(\mathbf{1}_{z<0} - 2\mathbf{1}_{x_3<0}) + x_2\right] \partial z && (17)\\
&\approx \int_{z=0}^{x_s} -2\pi\cos(2\pi z)\mathbf{1}_{z<0} && (18)\\
&\approx \sin(2\pi x_1)\mathbf{1}_{x_1<0} && (19)
\end{aligned}
$$

The result is shown in Figure 2(Left). Furthermore, RHALE visualizes the heterogeneity as $\mu(x_1) \pm H_1^{\texttt{RHALE}}(x_1)$, where $\mu(x_1)$ is the derivative effect and $H_1^{\texttt{RHALE}}(x_1)$ is the heterogeneity. These are given by:

$$
\begin{aligned}
\mu(x_1) &= \mathbb{E}_{x_2,x_3|x_1}\left[\frac{\partial f}{\partial x_1}(x_1, x_2, x_3)\right] && (20)\\
&= -2\pi\cos(2\pi x_1)\mathbf{1}_{x_1<0} && (21)
\end{aligned}
$$

$$
\begin{aligned}
H_1^{\texttt{RHALE}}(x_1) &= \sigma_{x_2,x_3|x_1}\left[\frac{\partial f}{\partial x_1}(x_1, x_2, x_3)\right] && (22)\\
&= \sigma_{x_2,x_3|x_1}\left[2\pi\cos(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) + x_2\right] && (23)\\
&= \sigma_2 = 2 && (24)
\end{aligned}
$$

The heterogeneity informs that the instance-level effects are deviating from the average derivative-effect by ± 2, as shown in Figure 2.

Approximation using `Effector`. Listing 2 shows how to use `Effector` to approximate the ALE effect for feature x_1 using K bins. The results are shown in Figure 3; at the left figure we observe the ALE effect with 5 bins, and in the right figure, we observe the ALE effect with 20 bins. At the top of the figures, we observe the global effect, and at the bottom, the derivative effect along with the heterogeneity visualized with vertical red bars. We should observe that both $K = 5$ and $K = 20$ bins lead to weak approximations of the ALE effect. This is because ALE's approximation is sensitive to the choice of K and in specific cases, no fixed-size binning can provide a good approximation of the ALE effect.

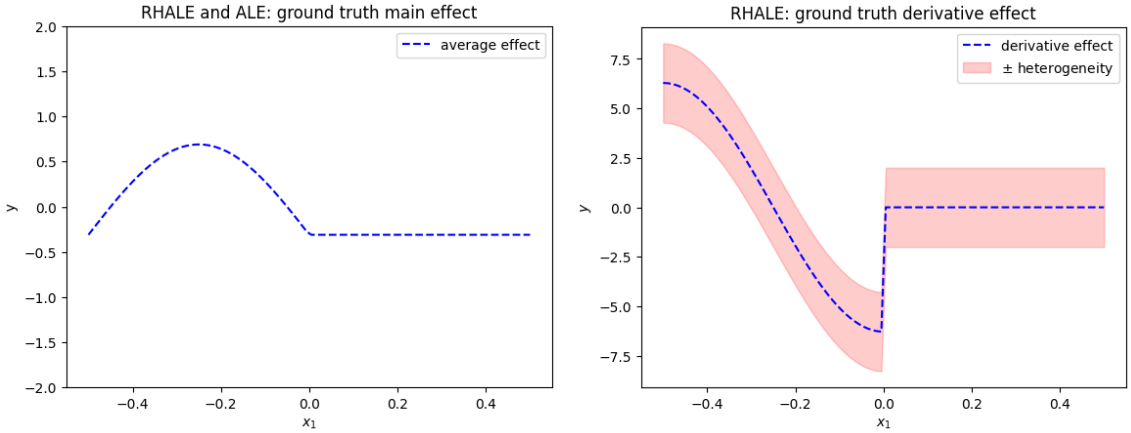


Figure 2: Left: ALE and RHALE global effect for feature x_1 based on their *definitions*. Right: RHALE derivative effect $\mu(x_1)$ and its heterogeneity $H_1^{\texttt{RHALE}}(x_1)$ visualized as $\mu(x_1) \pm H_1^{\texttt{RHALE}}(x_1)$.

```python
from effector.binning_methods import Fixed
from effector import ALE

# input
# x = .., the dataset (np.array of shape=(N,D))
# f = .., predictive function (python callable)

# ALE
ale = effector.ALE(data=x, model=f)
for k in [5, 20]:
  binning_method = binning_methods.Fixed(nof_bins=K)
  ale.fit(feat, binning_method=binning_method)
  ale.plot(feature=0, heterogeneity=True, centering=True)
```

Listing 2: Listing with how to use `Effector` to approximate the ALE effect for the first feature using K bins.

In Listing 3, we show how to use `Effector` to approximate the RHALE effect for feature x_1 using `DynamicProgramming` and `Greedy` optimization. As stated above, RHALE automatically splits the s-th feature into variable-size bins. The automatic bin-splitting is performed by solving an optimization problem that minimizes the heterogeneity of the derivative effect within each bin. For an in-depth explanation of the optimization problem, we refer the reader to (Gkolemis et al., 2023a). The optimization problem can be solved either using `DynamicProgramming`, which ensures a globally optimal solution, or with a `Greedy` approach, which is faster but less accurate. `Effector` provides both alternatives, as shown in Listing 3. The outputs are shown in Figure 4. We observe that the RHALE approximation using `DynamicProgramming` (left) is more accurate than the one using `Greedy` (right), but both of them are more accurate than the fixed-size binning of ALE (Figure 3).

```python
from effector.binning_methods import DynamicProgramming, Greedy
```

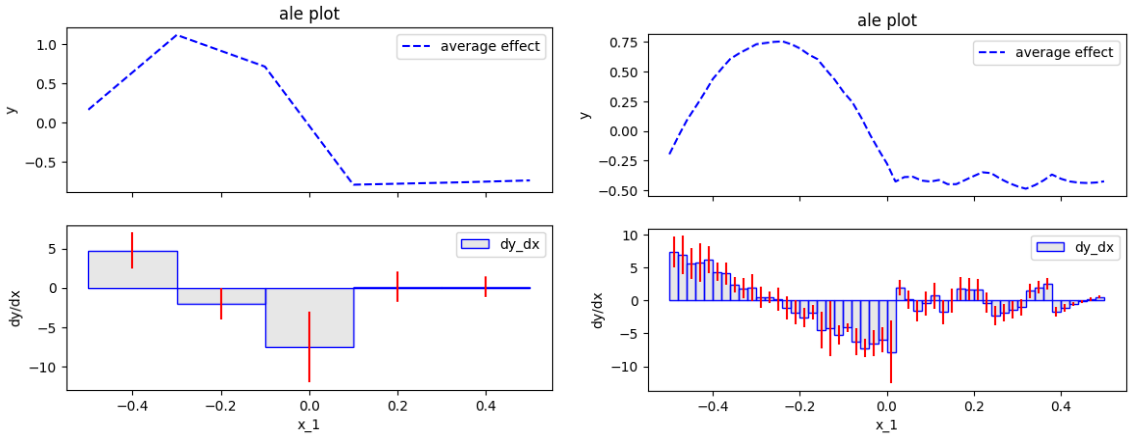


Figure 3: ALE effect for feature x_1 using 5 (left) and 20 bins (right).

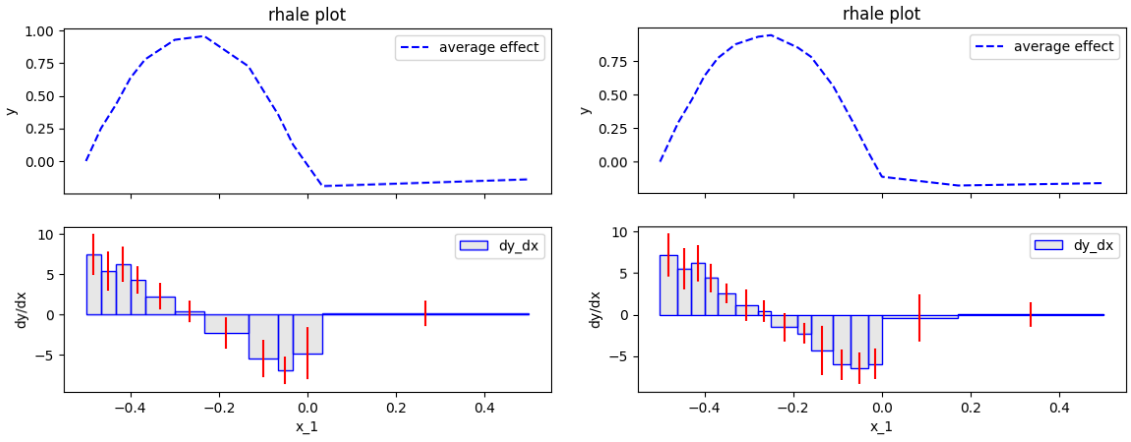


Figure 4: RHALE approximation using `DynamicProgramming` (left) and `Greedy` (right).

```
from effector import RHALE

# Input
# x = .., the dataset (np.array of shape=(N,D))
# f = .., predictive function (python callable)
# f_der = .., the model's derivative (python callable)

# RHALE
rhale = RHALE(data=x, model=f, model_jac=f_der)
binning = [
  Greedy(init_nof_bins=100, min_points_per_bin=10),
  DynamicProgramming(max_nof_bins=20, min_points_per_bin=10)
]

for binning_method in binning:
  rhale.fit(feat, binning_method=binning_method)
  rhale.plot(feature=0, heterogeneity=True, centering=True)
```

Listing 3: Demonstrating how to use `Effector` to approximate the RHALE effect for x_1 using `Dynamic Programming` and `Greedy` optimization.

C.1.3 PDP, D-PDP, ICE, D-ICE

Definition on the synthetic example: In the synthetic example, the PDP for feature x_1 is given by:

$$
\begin{aligned}
f^{\texttt{PDP}}(x_1) &= \mathbb{E}_{x_2,x_3}\left[f(x_1,x_2,x_3)\right] && (25)\\
&= \mathbb{E}_{x_2}\mathbb{E}_{x_3}\left[\sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) + x_1x_2 + x_2\right] && (26)\\
&= \sin(2\pi x_1)\mathbf{1}_{x_1<0} + \mathbb{E}_{x_3}\left[-2\sin(2\pi x_1)\mathbf{1}_{x_3<0}\right] + \mathbb{E}_{x_2}\left[x_1x_2 + x_2\right] && (27)\\
&= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbb{E}_{x_3}\left[\mathbf{1}_{x_3<0}\right)] && (28)\\
&= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\frac{5}{6}) && (29)\\
&= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - \frac{5}{3}) && (30)
\end{aligned}
$$

and the ICE plot for the i-th instance is given by:

$$
f_i^{\texttt{ICE}}(x_1) = \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3^i<0}) + x_1x_2^i \tag{31}
$$

The PDP-ICE is shown in Figure 5(Left). Additionally, the d-PDP for feature x_1 is given by:

$$
\begin{aligned}
f^{\texttt{d-PDP}}(x_1) &= \mathbb{E}_{x_2,x_3}\left[\frac{\partial f}{\partial x_1}(x_1,x_2,x_3)\right] && (32)\\
&= \mathbb{E}_{x_2}\mathbb{E}_{x_3}\left[2\pi\cos(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) + x_2\right] && (33)\\
&= 2\pi\cos(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\frac{5}{6}) && (34)
\end{aligned}
$$

and the d-ICE plot for the i-th instance is given by:

$$f_i^{\texttt{d-ICE}}(x_1) = 2\pi \cos(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3^i<0}) \tag{35}$$

which is shown in Figure 5(Right). We observe that the PDP-based methods treat the individual features as independent which leads to a non-zero effect for x_1 when $x_1 > 0$. Given the knowledge of the data-generating distribution, where $x_3 \approx x_1$, we can say that this effect is misleading.

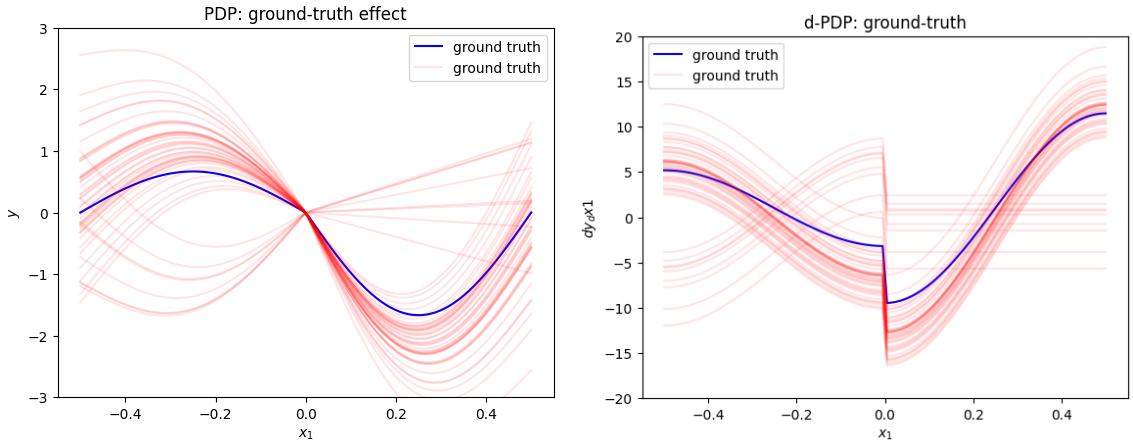


Figure 5: PDP-ICE (left) and d-PDP-ICE (right) definition on the synthetic example.

Approximation using `Effector`. Listing 4 shows how to use `Effector` to approximate the PDP and d-PDP effect for x_1. The outputs are shown in Figure 6. We observe that the PDP-ICE (left) and d-PDP-ICE (right) *approximations* are very close to the respective *definitions* (Figure 5).

```
from effector import PDP, dPDP

# Input
# x = .., the dataset (np.array of shape=(N,D))
# f = .., predictive function (python callable)

# PDP
PDP(data=x, model=f).plot(feature=0, centering=True,
    heterogeneity="ice")

# d-PDP
dPDP(data=x, model=f, nof_instances=100).plot(feature=0,
    centering=False, heterogeneity="ice")
```

Listing 4: Listing showing how to use `Effector` to approximate the PDP and d-PDP effect for x_1.

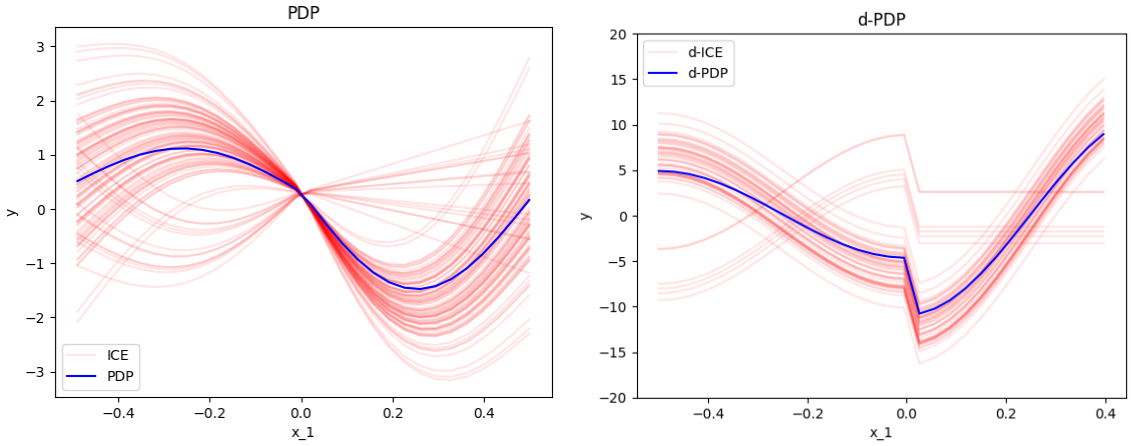


Figure 6: PDP-ICE approximation (left) and d-PDP-ICE approximation (right) on the synthetic example.

C.1.4 SHAP Dependence Plot

Definition on the synthetic example. Finding the SHAP-DP of the j-th feature of the i-th instance requires the computation of the value of all coalitions.

$$
\begin{aligned}
v(\emptyset) &= \mathbb{E}_{x_1,x_2,x_3}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] && (36)\\
&= -\frac{5}{6\pi} + 0 + 0 + 0 && (37)\\
&= -\frac{5}{6\pi} && (38)
\end{aligned}
$$

$$
\begin{aligned}
v(\{1\}) &= \mathbb{E}_{x_2,x_3}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] && (39)\\
&= \sin(2\pi x_1)\mathbf{1}_{x_1<0} - \frac{5}{3}\sin(2\pi x_1) + 0 + 0 && (40)\\
&= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - \frac{5}{3}) && (41)
\end{aligned}
$$

$$
\begin{aligned}
v(\{2\}) &= \mathbb{E}_{x_1,x_3}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] && (42)\\
&= -\frac{5}{6\pi} + 0 - \frac{1}{6}x_2 + x_2 && (43)\\
&= -\frac{5}{6\pi} + \frac{5}{6}x_2 && (44)
\end{aligned}
$$

$$v(\{3\}) = \mathbb{E}_{x_1,x_2}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] \tag{45}$$
$$= -\frac{5}{6\pi} + 0 + 0 + 0 \tag{46}$$
$$= -\frac{5}{6\pi} \tag{47}$$

$$v(\{1,2\}) = \mathbb{E}_{x_3}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] \tag{48}$$
$$= \sin(2\pi x_1)\mathbf{1}_{x_1<0} - \frac{5}{3}\sin(2\pi x_1) + x_1x_2 + x_2 \tag{49}$$

$$v(\{1,3\}) = \mathbb{E}_{x_2}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] \tag{50}$$
$$= \sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + 0 + 0 \tag{51}$$
$$= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) \tag{52}$$

$$v(\{2,3\}) = \mathbb{E}_{x_1}\left[\sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2\right] \tag{53}$$
$$= -\frac{5}{6\pi} + 0 - \frac{1}{6}x_2 + x_2 \tag{54}$$
$$= -\frac{5}{6\pi} + \frac{5}{6}x_2 \tag{55}$$

$$v(\{1,2,3\}) = \sin(2\pi x_1)\mathbf{1}_{x_1<0} - 2\sin(2\pi x_1)\mathbf{1}_{x_3<0} + x_1x_2 + x_2 \tag{56}$$

The contribution of feature x_1 when entering in any possible coalition is:

$$\Delta_{\emptyset,1} = v(\{1\}) - v(\emptyset) \tag{57}$$
$$= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - \frac{5}{3}) + \frac{5}{6\pi} \tag{58}$$

$$\Delta_{\{2\},1} = v(\{1,2\}) - v(\{2\}) \tag{59}$$
$$= \sin(2\pi x_1)\mathbf{1}_{x_1<0} - \frac{5}{3}\sin(2\pi x_1) + x_1x_2 + \frac{1}{6}x_2 + \frac{5}{6\pi} \tag{60}$$

$$\Delta_{\{3\},1} = v(\{1,3\}) - v(\{3\}) \tag{61}$$
$$= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0}) + \frac{5}{6\pi} \tag{62}$$

$$\Delta_{\{2,3\},1} = v(\{1,2,3\}) - v(\{2,3\}) \tag{63}$$

$$= \sin(2\pi x_1)(\mathbf{1}_{x_1<0} - 2\mathbf{1}_{x_3<0})) + x_1 x_2 + \frac{1}{6}x_2 + \frac{5}{6\pi} \tag{64}$$

Therefore, the SHAP value of the first feature of a particular instance i is:

$$\phi_1^i = \frac{1}{3}\Delta_{\{0\},1} + \frac{1}{6}\Delta_{\{2\},1} + \frac{1}{6}\Delta_{\{3\},1} + \frac{1}{3}\Delta_{\{2,3\},1} \tag{65}$$

$$= \sin(2\pi x_1)\mathbf{1}_{x_1<0} - \sin(2\pi x_1)\mathbf{1}_{x_3<0} - \frac{5}{6}\sin(2\pi x_1) + \frac{1}{2}x_1 x_2 + \frac{1}{32}x_2 + \frac{5}{6\pi} \tag{66}$$

$$\approx -\frac{5}{6}\sin(2\pi x_1) + \frac{1}{2}x_1 x_2 + \frac{1}{32}x_2 + \frac{5}{6\pi} \tag{67}$$

and the SHAP-DP, $f^{\texttt{SHAP-DP}}(x_1)$, is given by a one-dimensional spline fit to $\{x_s^i, \hat{\phi}_s^i\}_{i=1}^N$ as shown in Figure 7 (left).

Approximation using `Effector`. In Listing 5, we show how to use `Effector` to approximate the SHAP-DP for the first feature. In Figure 7, we observe that the SHAP-DP *approximation* is very close to the *definition*. As was the case with the PDP-based plots, SHAP-DP also treats the individual features as independent, leading to a non-zero effect for x_1 when $x_1 > 0$.

```python
from effector import SHAPDependence

# Input
# x = .., the dataset (np.array of shape=(N,D))
# f = .., predictive function (python callable)

# SHAP-DP
SHAPDependence(data=x, model=f).plot(feature=0, centering=True,
    heterogeneity="shap_values")
```

Listing 5: Listing showing how to use `Effector` to approximate the SHAP-DP for x_1.

C.1.5 Conclusion

In this illustration, we showcase how `Effector` is utilized to generate global effect plots. The example highlights the simplicity of `Effector`'s API, typically requiring just one or two lines of code for plot generation. Additionally, the consistency of the API across various methods facilitates users in effortlessly comparing the outputs of different methods. As demonstrated, within the realm of global effect plots, there is no singular ground-truth effect. Hence, users typically seek to compare the effects obtained from different methods.

We also show that under highly-correlated features, ALE-based methodologies emerge as more reliable. In the provided example, only ALE-based techniques accurately depict that the effect of x_1 on the output diminishes to zero for $x_1 > 0$. Conversely, other methods depict a sinusoidal effect for $x_1 > 0$ due to their implicit assumption of feature independence.

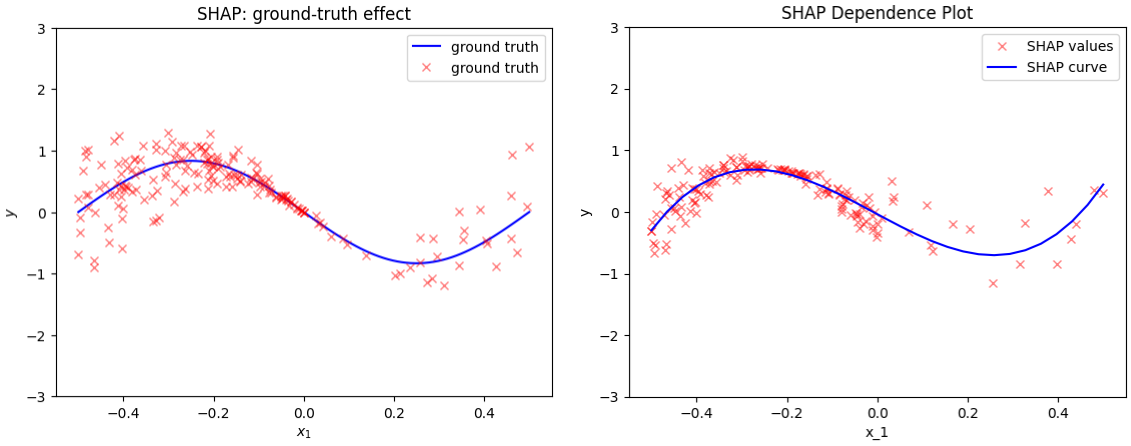


Figure 7: SHAP Dependence Plot definition (left) and approximation (right).

Additionally, we emphasize that within ALE-based frameworks, RHALE emerges as a faster and more precise approximation of the ALE definition, leveraging automatic-differentiation for speed and automatic bin-splitting for precision.

C.2 Regional Effect: Comparative study of `Effector`'s methods

This example shows how to use `Effector` for generating regional effect plots. Regional effect plots are a powerful tool for visualizing the effect of a feature on the output of a model across different regions of the feature space.

C.2.1 Problem set-up

The data generating distribution is $X_i \sim p(x_i) = \mathcal{U}(x_i; -1, 1)$, for $i = \{1, 2, 3\}$ and the 'black-box' function is $f(x) = 3x_1 I_{x_3>0} - 3x_1 I_{x_3\leq 0} + x_3$. The terms $3x_1 I_{x_3>0}$ and $3x_1 I_{x_3\leq 0}$ are the ones that provoke heterogeneity in the effect of x_1. It is immediate to see that if splitting the feature space into two regions, one where $x_3 > 0$ and another where $x_3 \leq 0$, the heterogeneity in the effect of x_1 becomes zero.

C.3 ALE and RHALE

Listing 6 shows how to use `Effector` to obtain regional effects using RHALE. For obtaining regional ALE effects, the only difference would be using `RegionalALE` instead of `RegionalRHALE`. At line 18, we define that an additional split is only allowed if the heterogeneity drops over 60% (0.6). At line 20, we define that the number of candidate splits for numerical features is 11. After fitting the model, we can visualize the partitioning of the feature space at line 23 using the method `.show_partitioning()`. The printed output informs that the feature space is split into two regions, $x_3 \leq 0$ with 496 and $x_3 > 0$ with 504 instances, respectively. The split provokes a heterogeneity drop of 100%, from 5.94 to 0.0. Finally, we can visualize the global RHALE effect at line 34 and the regional effects at lines 37 and 38. The outputs are shown in Figure 8.

For features x_2 and x_3, the algorithm does not split in subspaces because the heterogeneity of the global effect is already zero. The outputs are shown in Figure 9.

```
from effector import RegionalRHALE
from effector.binning_methods import Fixed

# Input
# x = .., the dataset (np.array of shape=(N,D))
# f = .., predictive function (python callable)
# f_der = .., the model's derivative (python callable)

regional_rhale = effector.RegionalRHALE(
    data=x,
    model=f,
    model_jac=f_der
)

binning_method = Fixed(11, min_points_per_bin=0)
regional_rhale.fit(
    features="all",
    heter_pcg_drop_thres=0.6,
    binning_method=binning_method,
    nof_candidate_splits_for_numerical=11
)

regional_rhale.show_partitioning(features=0)
# Feature 0 - Full partition tree:
# Node id: 0, name: x1, heter: 5.94 || nof_instances:  1000 || weight: 1.00
#         Node id: 1, name: x1 | x3 <= -0.0, heter: 0.00 || nof_instances:   496 || weight: 0.50
#         Node id: 2, name: x1 | x3  > -0.0, heter: 0.00 || nof_instances:   504 || weight: 0.50
# ----------------------------------------------------
# Feature 0 - Statistics per tree level:
# Level 0, heter: 5.94
#         Level 1, heter: 0.00 || heter drop: 5.94 (100.00%)

# global effect
regional_rhale.plot(feature=0, node_idx=0, heterogeneity="std", centering=True)

# regional effect
regional_rhale.plot(feature=0, node_idx=1, heterogeneity="std", centering=True)
regional_rhale.plot(feature=0, node_idx=2, heterogeneity="std", centering=True)
```

Listing 6: Listing showing how to use `Effector` for obtaning regional effects using RHALE.

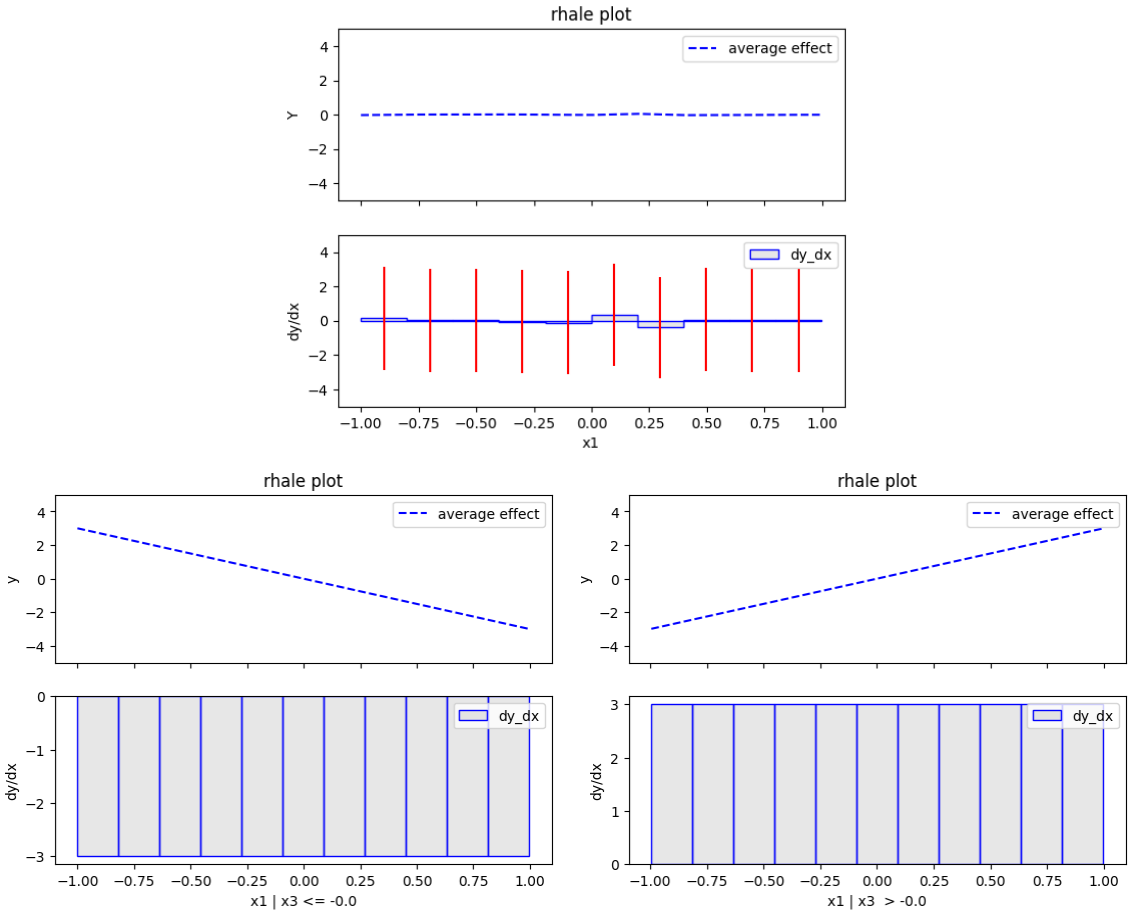


Figure 8: Top: Global RHALE effect for feature x_1. Bottom: Regional RHALE effect for feature x_1 in the regions $x_3 \leq 0$ (left) and $x_3 > 0$ (right).

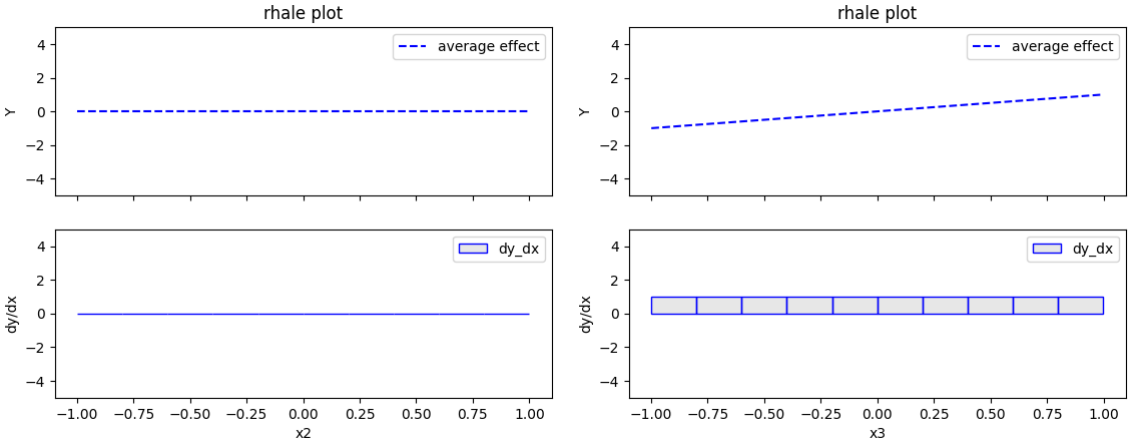


Figure 9: Global RHALE effect for feature x_2 (left) and x_3 (right).

C.4 PDP-ICE

Listing 6 shows how to use `Effector` to obtain regional effects using PDP-ICE. The process is identical for obtaining regional effects with d-PDP and d-ICE, with the only difference being that the method used would be `RegionalDerivativePDP` instead of `RegionalPDP`. In line 11, we define that an additional split is only allowed if the heterogeneity drops over 30% (0.3). In line 12, we define that the number of candidate splits for numerical features is 11. After fitting the model, we can visualize the partitioning of the feature space at line 15. The printed output informs that the feature space is split into two regions, $x_3 \leq 0$ with 496 and $x_3 > 0$ with 504 instances, respectively. The split provokes a heterogeneity drop of 83.59%, from 1.74 to 0.29. Finally, we can visualize the global PDP effect at line 36 and the regional effects at lines 39 and 40. The outputs are shown in Figure 10.

It is interesting to notice that apart from x_1 the PDP-ICE method finds subspaces for x_3. As shown in lines 25-32 of Listing 4, the algorithm finds that by splitting in $x_1 \leq 0$ and $x_1 > 0$, the heterogeneity of the global effect of x_3 drops from 1.76 to 0.86, a drop of 51.24%. The outputs are shown in Figure 11. This happens because due to PDP formula there is an offset of $\pm 6x_1^i$ at $x_1 = 0$. In the subspace $x_1 \leq 0$, the offset is $-6x_1^i$ and in the subspace $x_1 > 0$, the offset is $6x_1^i$. For feature x_2 and x_3, the algorithm does not split in subspaces.

```
1  from effector import RegionalPDP
2
3  # Input
4  # x = .., the dataset (np.array of shape=(N,D))
5  # f = .., predictive function (python callable)
6  # f_der = .., the model's derivative (python callable)
7
8  regional_pdp = effector.RegionalPDP(data=x, model=f)
9  regional_pdp.fit(
10     features="all",
11     heter_pcg_drop_thres=0.3,
12     nof_candidate_splits_for_numerical=11
13 )
14
15 [regional_pdp.show_partitioning(features=i) for i in [0, 2]]
16 # Feature 0 - Full partition tree:
17 # Node id: 0, name: x1, heter: 1.74 || nof_instances:  1000 ||
     weight: 1.00
18 #         Node id: 1, name: x1 | x3 <= -0.0, heter: 0.28 ||
     nof_instances:   496 || weight: 0.50
19 #         Node id: 2, name: x1 | x3  > -0.0, heter: 0.29 ||
     nof_instances:   504 || weight: 0.50
20 # ----------------------------------------------------
21 # Feature 0 - Statistics per tree level:
22 # Level 0, heter: 1.74
23 #         Level 1, heter: 0.29 || heter drop: 1.45 (83.59%)
24
25 # Feature 2 - Full partition tree:
26 # Node id: 0, name: x3, heter: 1.76 || nof_instances:  1000 ||
     weight: 1.00
```

```
#          Node id: 1, name: x3 | x1 <= -0.0, heter: 0.85 || nof_instances:   507 || weight: 0.51
#          Node id: 2, name: x3 | x1  > -0.0, heter: 0.87 || nof_instances:   493 || weight: 0.49
# ----------------------------------------------------
# Feature 2 - Statistics per tree level:
# Level 0, heter: 1.76
# Level 1, heter: 0.86 || heter drop: 0.90 (51.24%)

for i in [0, 1]:
    # Global effect
    regional_pdp.plot(feature=0, node_idx=0, heterogeneity="ice", centering=True)

    # regional effect
    regional_pdp.plot(feature=0, node_idx=1, heterogeneity="ice", centering=True)
    regional_pdp.plot(feature=0, node_idx=2, heterogeneity="ice", centering=True)
```

Listing 7: Listing showing how to use `Effector` for obtaning regional effects using PDP-ICE.

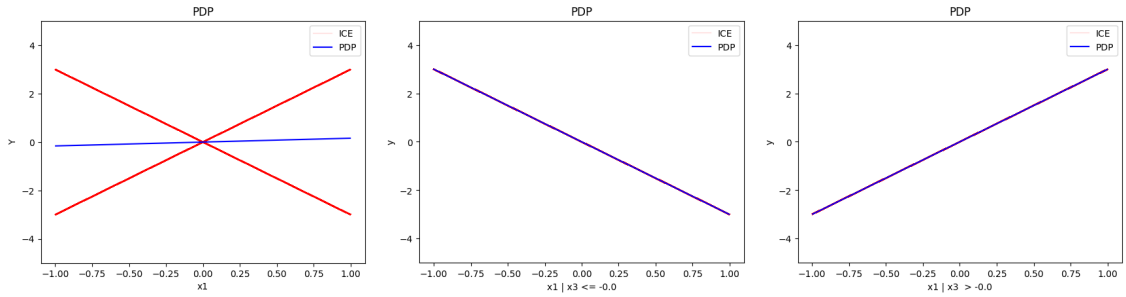


Figure 10: Left: Global PDP effect for feature x_1. Middle and Right: Regional PDP effect for feature x_1 in the regions $x_3 \leq 0$ (left) and $x_3 > 0$

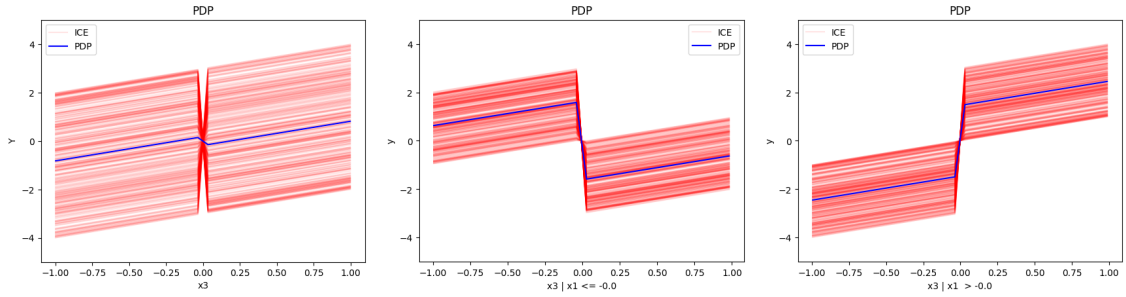


Figure 11: Left: Global PDP effect for feature x_3. Middle and Right: Regional PDP effect for feature x_3 in the regions $x_1 \leq 0$ (left) and $x_1 > 0$.

C.5 SHAP-DP

Listing 8 shows how to use `Effector` to obtain regional effects using RHALE. The process is identical for obtaining regional ALE effects, with the only difference being that the method used would be `RegionalALE` instead of `RegionalRHALE`. At line 18, we define that an additional split is only allowed if the heterogeneity drops over 60% (0.6). In line 20, we define that the number of candidate splits for numerical features is 11. After fitting the model, we can visualize the partitioning of the feature space at line 22. The printed output informs that the feature space is split into two regions, $x_3 \leq 0$ with 496 and $x_3 > 0$ with 504 instances, respectively. The split provokes a heterogeneity drop of 100%, from 5.94 to 0.0. Finally, we can visualize the global ALE effect at line 24 and the regional effects at lines 24 and 25. The outputs are shown in Figure 12.

For features x_2 and x_3, the algorithm does not split into subspaces. For x_2 this happens because the heterogeneity of the global effect is already zero. For x_3 there is some heterogeneity in the global effect, but there is no split that can reduce it over 60%. The outputs are shown in Figure 13.

```
1  from effector import RegionalRHALE
2  from effector.binning_methods import Fixed
3
4  # Input
5  # x = .., the dataset (np.array of shape=(N,D))
6  # f = .., predictive function (python callable)
7
8  regional_shap = effector.RegionalSHAP(data=x, model=f)
9
10 regional_shap.fit(
11     features="all",
12     heter_pcg_drop_thres=0.6,
13     nof_candidate_splits_for_numerical=11
14 )
15
16 regional_shap.show_partitioning(features=0)
17 # Feature 0 - Full partition tree:
18 # Node id: 0, name: x1, heter: 0.80 || nof_instances:   100 ||
     weight: 1.00
19 #         Node id: 1, name: x1 | x3 <= -0.0, heter: 0.00 ||
     nof_instances:    49 || weight: 0.49
20 #         Node id: 2, name: x1 | x3  > -0.0, heter: 0.00 ||
     nof_instances:    51 || weight: 0.51
21 # ----------------------------------------------------
22 # Feature 0 - Statistics per tree level:
23 # Level 0, heter: 0.80
24 #         Level 1, heter: 0.00 || heter drop: 0.80 (100.00%)
25
26 # global effect
27 regional_shap.plot(feature=0, node_idx=0,
     heterogeneity="shap_values", centering=True)
28
29 # regional effect
```

```
regional_shap.plot(feature=0, node_idx=1,
    heterogeneity="shap_values", centering=True)
regional_shap.plot(feature=0, node_idx=2,
    heterogeneity="shap_values", centering=True)
```

Listing 8: Listing showing how to use `Effector` for obtaning regional effects using SHAP-DP.

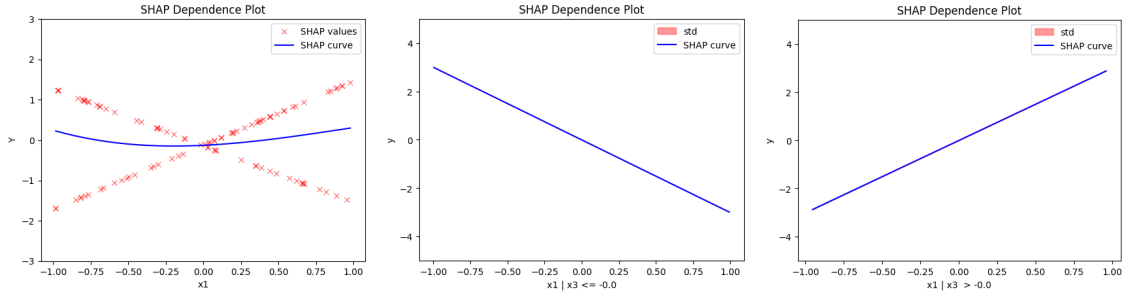


Figure 12: Left: Global SHAP-DP for feature x_1. Middle and Right: Regional SHAP-DP effect for feature x_1 in the regions $x_3 \leq 0$ (left) and $x_3 > 0$.

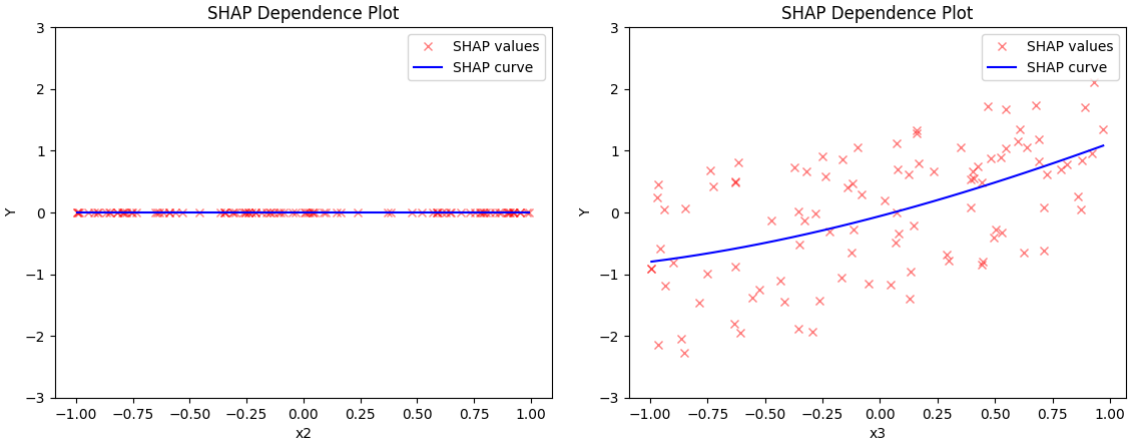


Figure 13: Global SHAP-DP for feature x_2 (left) and x_3 (right).

Appendix D. Real Example

In this section, we illustrate the use of `Effector` on a real-world dataset.

D.1 The Bike-Sharing dataset

The Bike-Sharing Dataset contains the hourly and daily count of rental bikes between the years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information.

Preprocessing and Model fitting. The dataset contains 14 features with information about the day-type, e.g., month, hour, which day of the week, whether it is working-day, and the weather conditions, e.g., temperature, humidity, wind speed, etc. The target variable is the number of bike rentals per hour. The dataset contains 17,379 instances.

We keep 11 features; $X_{\texttt{season}}$ (1: winter, 2: spring, 3: summer, 4: fall), $X_{\texttt{yr}}$ (0: 2011, 1: 2012), $X_{\texttt{mnth}}$ (1 to 12), $X_{\texttt{hr}}$ (0 to 23), $X_{\texttt{holiday}}$ (0: no, 1: yes), $X_{\texttt{weekday}}$ (0 to 6), $X_{\texttt{workingday}}$ (0: no, 1: yes), $X_{\texttt{weathersit}}$ (1: clear, 2: mist, 3: light rain, 4: heavy rain), $X_{\texttt{temp}}$ (normalized temperature), $X_{\texttt{hum}}$ (normalized humidity), and $X_{\texttt{windspeed}}$ (normalized wind speed). The target variable $Y_{\texttt{cnt}}$ is the number of bike rentals per hour, which ranges from 1 to 977, with a mean of 189.5 and a standard deviation of 181.4.

We fit a fully connected neural network with 3 hidden layers of 1024, 512, and 256 neurons, respectively, using the TensorFlow library. We use the Mean Squared Error as the loss function and the Adam optimizer with a learning rate of 0.001. We train the model for 20 epochs with a batch size of 512.

Global effect plots. In Figure 15, we illustrate the PDP-ICE and Figure 14 the RHALE effect for all features. The plots reveal a high degree of concordance between the PDP and RHALE explanations across all features. Some minor differences are the following. $X_{\texttt{year}}$ exhibits a more pronounced positive effect in RHALE compared to PDP when transitioning from 2011 to 2012. Conversely, for $X_{\texttt{holiday}}$, the RHALE effect appears more negative than in PDP when shifting from a non-holiday to a holiday. Finally, for $X_{\texttt{workingday}} = 1$, the effect appears more positive in RHALE than in PDP.

Of particular interest is the observation that among all features, the hour of the day ($X_{\texttt{hr}}$) exerts by far the most significant impact on bike rental numbers. The PDP and RHALE effect of $X_{\texttt{hr}}$ shows two peaks, occurring at 8:00 and 17:00, respectively. This pattern aligns with the typical usage of bikes for commuting to and from work. Subsequently, we delve deeper into the analysis of $X_{\texttt{hr}}$ utilizing regional effect plots.

Regional effect plots. We further analyze the effect of the hour of the day ($X_{\texttt{hr}}$) on bike rentals using regional effect plots. In Figure 17, we illustrate the PDP-ICE effect for $X_{\texttt{hr}}$ on non-working days and working days. The plots reveal a significant difference in the effect of $X_{\texttt{hr}}$ on bike rentals between non-working and working days. On non-working days, the effect of $X_{\texttt{hr}}$ on bike rentals peaks at 14:00, while on working days, the effect peaks at 8:00 and 17:00. This pattern aligns with the typical usage of bikes for leisure on non-working days and commuting to and from work on working days.

Appendix E. Integration with common Machine Learning libraries

`Effector` is designed to be easily integrated with common machine learning libraries such as `scikit-learn`, `TensorFlow` and `PyTorch`. The user only needs to provide a predictive function and its derivative, if available, to use `Effector` with a model trained using these libraries.

In this section, we show how to use `Effector` with a model trained using `scikit-learn` (Listing 9), `TensorFlow` (Listing 10), and `PyTorch` (Listing 11). In all cases, we train a Fully Connected Neural Network (DNN) with two hidden layers of 10 neurons each. We, then, use `RegionalRHALE` to obtain regional effects. Note that `scikit-learn` does not

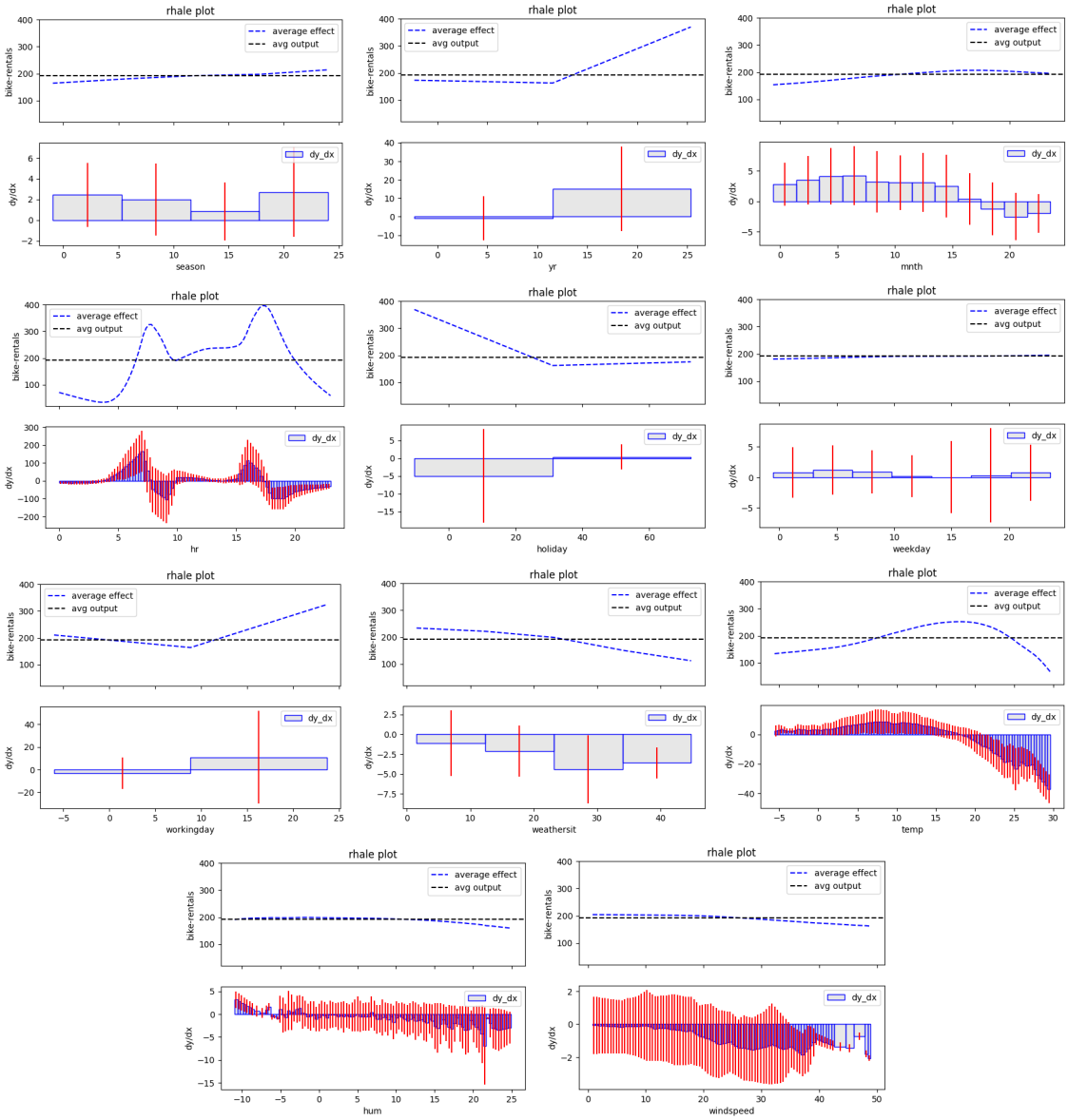


Figure 14: RHALE effect for all the features.

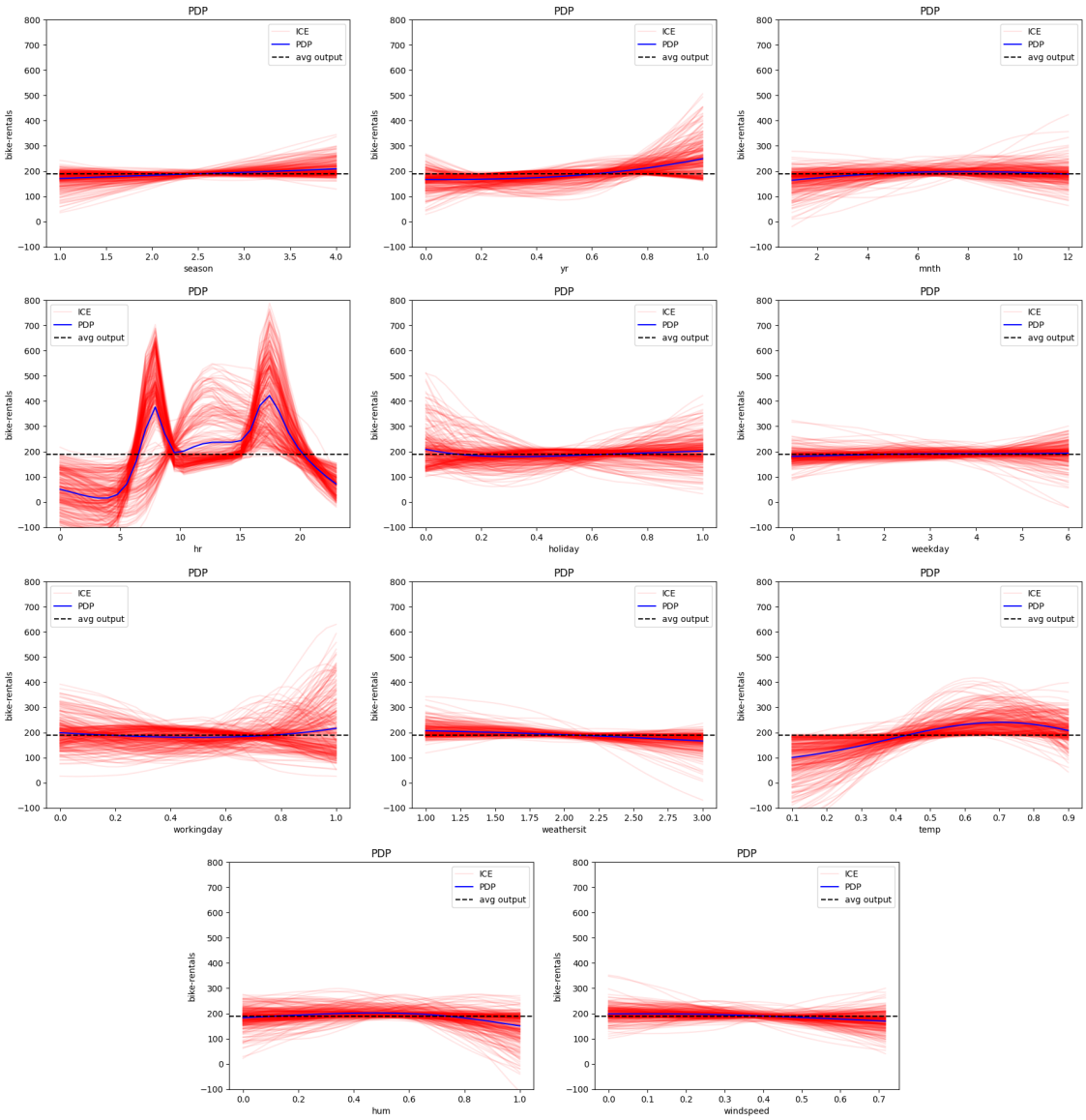


Figure 15: PDP-ICE effect for all the features.

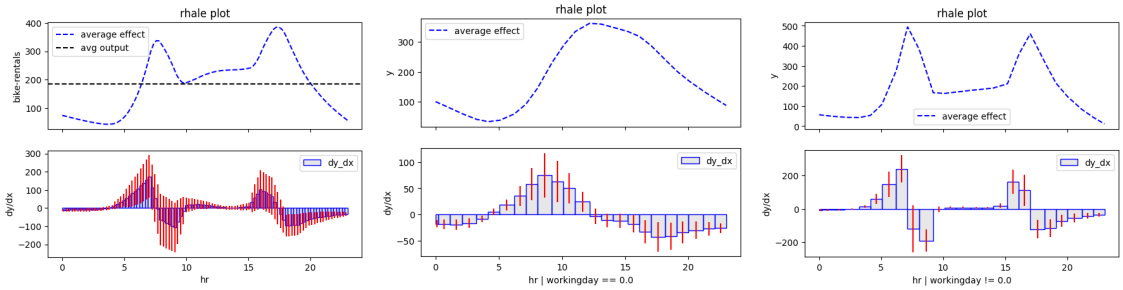


Figure 16: Top: Global RHALE effect for the temperature. Bottom: Regional RHALE effect for the temperature on non-working days (left) and working days (right).

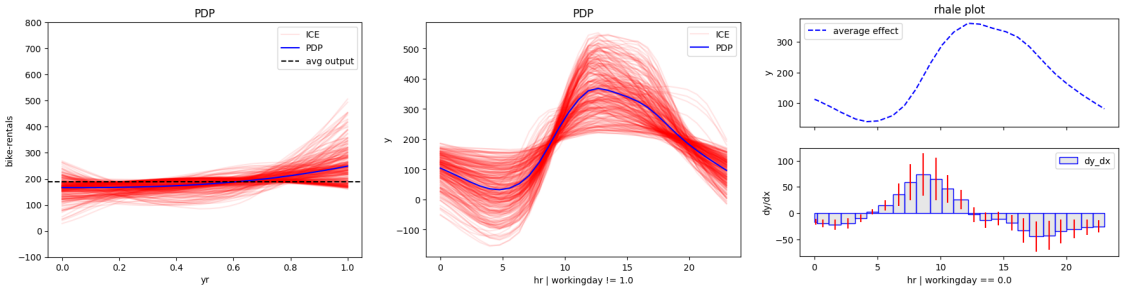


Figure 17: Top: Global PDP effect for the temperature. Bottom: Regional PDP effect for the temperature on non-working days (left) and working days (right).

provide auto-differentiation, so we only use the predictive function. On the other hand, `TensorFlow` and `PyTorch` provide auto-differentiation, so we use the predictive function and its derivative.

```
from sklearn.neural_network import MLPRegressor
from effector import RegionalRHALE

model = MLPRegressor(
    solver='adam',
    learning_rate='constant',
    hidden_layer_sizes=(10, 10),
    learning_rate_init=0.01,
    max_iter=100
)

model.fit(X, y)

def predict(X):
  return model.predict(X)

# scikit-learn does not provide auto-differentiation.
RegionalRHALE(data=X, model=predict).plot(feature=0, node_idx=0)
```

Listing 9: Integration of `Effector` with `scikit-learn`.

```python
import tensorflow as tf

# define model
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(1)
])

# train model
model.compile(optimizer='adam', loss='mse')
model.fit(X, y, epochs=10)

def predict(X: np.ndarray) -> np.ndarray:
  return model.predict(X)

def jaobian(X: np.ndarray) -> np.ndarray:
    X = tf.convert_to_tensor(X, dtype=tf.float32)
    with tf.GradientTape() as tape:
        tape.watch(X)
        y = model(X)
    return tape.gradient(y, X).numpy()

RegionalRHALE(X, predict, jacobian).plot(feature=0, node_idx=0)
```

Listing 10: Integration of `Effector` with `Tensorflow`.

```python
from torch import nn, optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(D, 10)
        self.fc2 = nn.Linear(10, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        return x

model = Net()

# train model
optimizer = optim.Adam(model.parameters(), lr=0.01)
criterion = nn.MSELoss()
for epoch in range(10):
  optimizer.zero_grad()
  y_pred = model(X)
  loss = criterion(y_pred, y)
  loss.backward()
  optimizer.step()

```

```python
def predict(X):
    return model(torch.tensor(X,
        dtype=torch.float32)).detach().numpy()

def jacobian(X):
    X = torch.tensor(X, dtype=torch.float32)
    X.requires_grad = True
    y = model(X)
    return torch.autograd.grad(y, X)[0].numpy()

RegionalRHALE(X, predict, jacobian).plot(feature=0, node_idx=0)
```

Listing 11: Integration of `Effector` with `PyTorch`.

References

R. Agarwal, L. Melnick, N. Frosst, X. Zhang, B. Lengerich, R. Caruana, and G. Hinton. Neural Additive Models: Interpretable Machine Learning with Neural Nets, Oct. 2021. URL http://arxiv.org/abs/2004.13912. arXiv:2004.13912 [cs, stat].

D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020. Publisher: Wiley Online Library.

H. Baniecki, W. Kretowicz, P. Piatyszek, J. Wiśniewski, and P. Biecek. dalex: Responsible Machine Learning with Interactive Explainability and Fairness in Python. *Journal of Machine Learning Research*, 22(214):1–7, 2021. URL http://jmlr.org/papers/v22/20-1473.html.

M. Britton. Vine: Visualizing statistical interactions in black box models. *arXiv preprint arXiv:1904.00561*, 2019.

H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2:113–127, 2014.

T. Freiesleben, G. König, C. Molnar, and A. Tejero-Cantero. Scientific Inference With Interpretable Machine Learning: Analyzing Models to Learn About Real-World Phenomena. URL http://arxiv.org/abs/2206.05487.

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

J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *The annals of applied statistics*, pages 916–954, 2008. Publisher: JSTOR.

V. Gkolemis, T. Dalamagas, and C. Diou. Dale: Differential accumulated local effects for efficient and accurate global explanations. In *Asian Conference on Machine Learning (ACML)*, Oct. 2022.

V. Gkolemis, T. Dalamagas, E. Ntoutsi, and C. Diou. Rhale: Robust and heterogeneity-aware accumulated local effects. In *ECAI 2023*, pages 859–866. IOS Press, 2023a.

V. Gkolemis, A. Tzerefos, T. Dalamagas, E. Ntoutsi, and C. Diou. Regionally additive models: Explainable-by-design models minimizing feature interactions. *arXiv preprint arXiv:2309.12215*, 2023b.

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation, Mar. 2014. URL http://arxiv.org/abs/1309.6392. arXiv:1309.6392 [stat].

J. Herbinger, B. Bischl, and G. Casalicchio. REPID: Regional Effect Plots with implicit Interaction Detection, Feb. 2022. URL http://arxiv.org/abs/2202.07254. arXiv:2202.07254 [cs, stat].

J. Herbinger, B. Bischl, and G. Casalicchio. Decomposing global feature effects based on feature interactions. *arXiv preprint arXiv:2306.00541*, 2023.

L. Hu, J. Chen, V. N. Nair, and A. Sudjianto. Surrogate locally-interpretable models with supervised machine learning algorithms. *arXiv preprint arXiv:2007.14528*, 2020.

J. Klaise, A. V. Looveren, G. Vacanti, and A. Coca. Alibi Explain: Algorithms for Explaining Machine Learning Models. *Journal of Machine Learning Research*, 22(181):1–7, 2021. URL http://jmlr.org/papers/v22/21-0017.html.

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

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021. Publisher: ACM New York, NY, USA.

C. Molnar, G. König, B. Bischl, and G. Casalicchio. Model-agnostic feature importance and effects with dependent features: a conditional subgroup approach. *Data Mining and Knowledge Discovery*, pages 1–39, 2023.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

M. T. Ribeiro, S. Singh, and C. Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

C. A. Scholbeck, G. Casalicchio, C. Molnar, B. Bischl, and C. Heumann. Marginal effects for non-linear prediction functions. *arXiv preprint arXiv:2201.08837*, 2022.

K. Sokol and P. Flach. Explainability fact sheets: A framework for systematic assessment of explainable approaches. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 56–67, 2020.