Exercise 3

Feature Effects

In this exercise, we are going to explore feature effects using Individual Conditional Expection, Partial Dependence Plots, and Accumulated Local Effects. After completing the tasks, you have a detailed understanding of how these methods work. Therefore, you can explain results better and get a feeling when which method is more suitable.

- You can get a bonus point for this exercise if you pass at least 85% of the tests. Code is automatically tested after pushing. If your tests fail, either your code is wrong, or you solved the task differently. In the second case, we will manually check your solution.
- Three collected bonus points result in a 0.33 increase of the final grade.
- You are allowed to work in groups with up to three people. You do not need to specify your ErgebnisPIN as it is already collected from the entrance test.
- Follow the 'README.md' for further instructions.
- Finish this exercise by 3rd November, 2021 at 11:59 pm.

General Note: The following tasks heavily rely on the lecture, and it is hardly possible to finish the tasks without the slides. Hence, it is recommended to complete this exercise sheet alongside reading the slides. Additionally to the tasks' description and the slides, functions' docstring provide code-specific information, which are worth reading.

1 Individual Conditional Expectation

Individual Conditional Expectation (ICE) plots display one line per instance that shows how the instance's prediction changes when a feature changes. The following tasks guide you to implement ICE from scratch.

Associated file: *tasks/ice.py*.

1.1 Calculation

Write the ICE algorithm following page 5 from the lecture. In particular, you have to integrate intervention and prediction in *calculate_ice* to finish the task.

1.2 Preparation

Use the function *calculate_ice* from the previous task to prepare the data for plotting. Iterate in *prepare_ice* over the number of rows s.t. you have as many curves as you have data points. Use the parameter *centered* to apply c-ICE. Adjust all y values s.t. the first y value starts at 0.

In both centered and uncentered cases, please note that it is necessary to sort the x values.

1.3 Plotting

After preparing the ICE data, we now want to plot it. Use all x and y values from *prepare_ice*. Set labels using *dataset* and plot all lines with an alpha (e.g. 0.2) value.

2 Partial Dependency Plot

The partial dependence plot (short PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model. A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic or more complex.

Associated file: *tasks/ice.py*.

2.1 Preparation

We now want to use ICE data to plot PDP. Call *calculate_ice* and iterate over the rows. Sort x_s again and average the corresponding y values. Follow slide 13 for more details.

2.2 Plotting

Use *prepare_pdp* to plot the data. Set the labels with *dataset* and plot the line with no transparency.

3 Accumulated Local Effects

Accumulated local effects describe how features influence the prediction of a machine learning model on average. ALE plots are a faster and unbiased alternative to PDPs, which you will, after solving the tasks, notice when running the tests.

Associated file: *tasks/ale.py*.

3.1 Binning

Since ALE works with bins, we have to select bounds first. Usually the bins are chosen s.t. every bin has roughly the same number of data points. However, to make it more simple, we use the same length for every interval.

Complete the function *get_bounds* to select bounds by using *n_intervals* on numpy's linspace. Those bounds are used for the ALE within the next tasks.

3.2 Calculation

Write the ALE algorithm from scratch inside the function *calculate_ale* using the bounds from the previous task and the help of slides 15ff. For each interval k:

- Select the observations inside the interval (if zero observations are found then return 0 for this interval). In general x_s can be counted as included if it is bigger than the lower bound and smaller equals the upper bound. However, make sure the elements in the first bound are included as well.
- Intervention: Use the relevant observations to create two new subsets:
  - *X_min*: Values at s-th feature position are replaced by the lower interval value z_{k-1}.
  - *X_max*‘: Values at s-th feature position are replaced by the higher interval value z_k.
- Prediction: Get the predictions from both *X_min* and *X_max*. Use the predictions to calculate a difference. Sum up the differences and divide by the number of observations in the interval to get a single value.

Use *np.cumsum* to perform the aggregation step and therefore to get accumulated values for each interval value. Finally, if the *center* parameter is set, center each interval by subtracting the average of all interval values

Hint: *zip(bounds, bounds[1:])* is a nice trick to iterate over the bounds.

3.3 Preparation

The function *calculate_ale* returns both the used bounds and the ALE data. To use those data with *matplotlib*, you have to get the corresponding x values using the bounds. Simply subtract the upper bound by the lower bound in the function *prepare_ale* to retrieve the centers.