Exercise 2

Interpretable Models

In this exercise, we are going to explore interpretable models. Linear models and decision trees are fitted and critically analyzed.

- 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. The code has to be pushed by everyone. Please mention your teammates' GitHub names in a separate file 'team.txt'.
- Mention your ErgebnisPIN from the iML lecture inside 'ergebnispin.txt'. Do not mention other information like name or matrikelnummer.
- Follow the 'README.md' for further instructions.
- Finish this exercise until 27th October 2021.

---

1 Plotting

Linear models can be explained by their weights, whereas decision trees can be analyzed by computing feature importance. To visualize both weights and feature importance, bar plots are a good choice.

Complete the function *plot_bar* in the file *plotting.py*. The function should display a bar diagram using x (labels) and y (values for the labels). Use *utils.styled_plot.plt*, which is a wrapper of *matplotlib.pyplot*.

2 Linear Models

In this section, we are fitting different linear models and plot their weights. Finally, a correlation analysis is done to check if the interpretation of the weights makes sense.

Complete the functions in the file *linear_models.py*.

2.1 Linear Regression

Fit a linear regression model from sklearn in the function *fit_linear_regression*. Use default hyperparameters and return the fitted model.

2.2 Custom Linear Regression

Using a sklearn model is straightforward but also hides the information on how the model is actually trained. Therefore, we want to implement a linear regression model exemplary by ourselves.

Fill in both *fit* and *predict* methods inside the class *MyLinearRegression*. Use the following equations below to solve the *fit* method:

$$\mathcal{L} = \mathcal{M}(X) - y, \tag{1}$$

$$\frac{\partial L}{\partial \theta} = \frac{X^T \cdot \mathcal{L}}{|y|}, \tag{2}$$

$$\frac{\partial L}{\partial b} = \frac{\sum \mathcal{L}}{|y|}, \tag{3}$$

$$\theta_{i+1} = \theta_i - \eta \frac{\partial L}{\partial \theta}, \tag{4}$$

$$b_{i+1} = b_i - \eta \frac{\partial L}{\partial b}. \tag{5}$$

The symbols have the following meaning:

- $\mathcal{M}$: Model.
- $\mathcal{M}(\cdot)$: Prediction function.
- $\mathcal{L}$: Loss.
- y: Ground truth data.
- $|y|$: Number of elements inside y.
- θ: Weights.
- b: Bias.
- η: Learning Rate.

2.3 Plot Weights

Use the function *plot_bar*, which you solved previously, to plot the linear weights of the linear regression models. Fill the function *plot_linear_regression_weights* for that. Have a look inside *utils.dataset.Dataset* to retrieve the x values and access the model's attributes to retrieve the corresponding values. Finally, return x and y.

2.4 Generalized Linear Model (GLM)

A linear regression might not be the best choice for the given dataset because a classification problem is given. Hence, we are looking for a linear model which naturally handles the multi-classification problem. Select and fit a suitable GLM from sklearn and complete the function *fit_generalized_linear_model*.

Hint: Although linear models do not initially support multi-classification problems, workarounds like One-Vs-Rest adapt the models to it.

2.5 Correlation Analysis

Since we now know how the weights look like and therefore what features are the most important, we want to find out if those weights are unambiguous. Perform a correlation analysis to check the correlations of a single feature. You can use pandas dataframe and access its correlation function.

Complete the function *correlation_analysis* and check the docstring for further information.

3 Decision Trees

In addition to linear models, a decision tree is yet another interpretable model. After fitting and plotting the feature importance, we are going to write the feature importance function by ourselves.

Complete the functions in the file *decision_trees.py*.

3.1 Decision Tree

In the function *fit_decision_tree*, you should fit a decision tree from sklearn and fit it with the training data.

3.2 Plot Feature Importance

Use the function *plot_bar* again, to complete *plot_feature_importance*. Instead of using the coefficients, you should access the tree's feature importances. Do not forget to set an appropriate y label.

3.3 Compute Feature Importances from Scratch

We now want to calculate the feature importance from scratch. Decision trees are designed s.t. the impurity ξ (a.k.a gini or entropy) is as low as possible. The feature importance is calculated by the decrease in node impurity weighted by the probability of reaching that node. That basically means features often used in the trees have higher importance than features less used.

Mathematically speaking, the feature importance FI_j for the j-th feature can be calculated the following way:

$$\mathrm{FI}_j = \frac{\sum_{i \in \mathrm{Nodes}_j} \xi_i \cdot \mathrm{samples}_i - \xi_{\mathrm{left}(i)} \cdot \mathrm{samples}_{\mathrm{left}(i)} - \xi_{\mathrm{right}(i)} \cdot \mathrm{samples}_{\mathrm{right}(i)}}{\mathrm{samples}_0}. \tag{6}$$

The symbols have the following meaning:

- Nodes_j: Node ids which use j-th feature as decision.
- $\mathrm{left}(i)$: Returns the left node id from the i-th node.
- $\mathrm{right}(i)$: Returns the right node id from the i-th node.
- $\mathrm{samples}_i$: Number of samples from the i-th node. The root node is referred to 0.

Fill the function *compute_feature_importance*, in which you have to iterate over *model.tree_.feature*.

3.4 Normalize Feature Importances

After FI_j has been calculated, the feature importance has to be normalized. Use the accumulated feature importance over all features and calculate the normalized feature importance:

$$\widetilde{\mathrm{FI}}_j = \frac{\mathrm{FI}_j}{\sum_{i \in \mathrm{Features}} \mathrm{FI}_i}. \tag{7}$$

Complete the function *normalize_feature_importance* and return the feature importance for all features. Confirm your results with *model.feature_importances_*.