0.0.1 XGBoost

Up to this point, the analysis has focused on parametric approaches; we now turn to Extreme Gradient Boosting (XGBoost), a non-parametric machine learning method.
Machine Learning (ML) is one of the many disciplines that constitute the foundation of most AI research, and it consists in adapting to new circumstances as well as detecting and extrapolating patterns [Russell and Norvig, 2021]. It is a field within Artificial Intelligence (AI) that allows software to improve prediction capabilities based on historical data without being explicitly programmed, by generating future outputs from the identification of patterns in past data [Mian et al., 2024]. Machine learning modeling typically relies on non-parametric methods, as it does not explicitly depend on parameters to capture the behavior of the underlying phenomenon [Parmezan et al., 2019]. A central challenge addressed by the field is how to enable computers to learn from data without explicit programming, with the objective of mimicking human intelligence. In many real-world scenarios, the relationships between variables are too complex to be described through a simple closed-form input–output expression. Machine learning methods address this issue by constructing mathematical models that automatically learn and represent these intricate interactions through data analysis, with the ultimate goal of optimizing performance with respect to a specific objective [Mian et al., 2024].
Without delving into details, it is important to note that machine learning can be broadly categorized into four types: supervised learning, typically applied to classification and regression tasks; unsupervised learning, mainly used for clustering; semi-supervised learning, which combines labeled and unlabeled data; and reinforcement learning, often applied to autonomous decision-making and control [Taeho, 2023]. Since our focus is on the development of XGBoost, we will be working within the framework of supervised learning algorithms, specifically applying it to regression tasks.

The main idea of boosting is to turn weak learners[1] into strong ones by repeatedly correcting their mistakes. In gradient boosting, this is done through gradient descent, which minimizes the residual errors. XGBoost, short for Extreme Gradient Boosting, pushes this approach further by optimizing both speed and accuracy. Thanks to its efficiency, it has become a dominant tool in Kaggle competitions, consistently outperforming other models [Wade, 2020].

Gradient Boosting and Additive Model
A general gradient descent boosting framework is designed to construct additive models under virtually any fitting criterion. In this approach, weak learners are sequentially trained so that each new model corrects the errors of its predecessors. In particular, gradient boosting with regression trees has proven to be a competitive, robust, and interpretable method for both regression and classification tasks, making it especially suitable for analyzing noisy or imperfect datasets [Friedman, 2001].
In the context of function estimation or predictive learning, the objective is to model the

[1]The weak learners considered in the implementation are shallow decision trees and linear models. By contrast, Dart and Random Forest are not weak learners but rather ensemble strategies; they are nevertheless included in the implementation for comparison purposes and to assess performance against alternative ensemble approaches.

relationship between a random output variable y and a set of random input (or explanatory) variables $\mathbf{x} = \{x_1, \ldots, x_n\}$. Given a training sample $\{(y_i, \mathbf{x}_i)\}_{i=1}^N$ of observed pairs, the goal is to construct an estimate $\hat{F}(\mathbf{x})$ of the true underlying function $F^*(\mathbf{x})$, which maps inputs $\mathbf{x}$ to outputs y. This estimation is carried out by minimizing the expected value of a chosen loss function $L(y, F(\mathbf{x}))$, which measures the discrepancy between the observed outcome y and the model's prediction $F(\mathbf{x})$. Formally, the optimal function is defined as [Friedman, 2001]:

$$F^* = \arg\min_F \mathbb{E}_{y,\mathbf{x}}\big[L(y, F(\mathbf{x}))\big] = \arg\min_F \mathbb{E}_{\mathbf{x}}\Big[\mathbb{E}_y\big(L(y, F(\mathbf{x})) \mid \mathbf{x}\big)\Big].$$

In practice, different choices of the loss function $L(y, F(\mathbf{x}))$ are employed depending on the type of problem. For regression tasks, common options include the *squared error loss* $(y - F(\mathbf{x}))^2$ and the *absolute error loss* $|y - F(\mathbf{x})|$. These functions provide the criterion according to which the learning algorithm adapts the model to approximate the true function $F^*(\mathbf{x})$ as accurately as possible [Friedman, 2001].
In our implementation, the XGBoost model was trained with the squared-error objective, aiming to minimize the Mean Squared Error (MSE) between predicted and actual values.
In practice, the estimation of $F(\mathbf{x})$ is typically restricted to a parameterized class of functions $F(\mathbf{x}; \mathbf{P})$, where $\mathbf{P} = \{P_1, P_2, \ldots\}$ denotes a finite set of parameters (e.g., in linear regression, the parameter set $\mathbf{P}$ corresponds to the regression coefficients $\boldsymbol{\beta}$). Each specific choice of parameters uniquely defines a function within this class [Friedman, 2001].
We now proceed to introduce a fundamental component of our framework, namely the additive model.
The equation below

$$F(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M) = \sum_{m=1}^{M} \beta_m \, h(\mathbf{x}; \mathbf{a}_m)$$

represents an *additive model*, where the function $F(\mathbf{x})$ is constructed as a weighted sum of simpler functions $h(\mathbf{x}; \mathbf{a}_m)$.

- The function $h(\mathbf{x}; \mathbf{a})$ acts as a *base learner*, typically chosen to be a simple, parameterized function of the input variables $\mathbf{x}$.
- The parameters $\mathbf{a} = \{a_1, a_2, \ldots\}$ describe the internal structure of the learner. For example, they may represent split points in a regression tree, or coefficients in a linear model.
- Each base learner is associated with a weight β_m, which regulates its relative contribution to the overall model.

In the most widely adopted case of *gradient boosting*, the base learner $h(\mathbf{x}; \mathbf{a}_m)$ is usually chosen to be a small regression tree[2]. Here, the parameters $\mathbf{a}_m$ define the splitting variables and the terminal node means of the tree. The boosted model is thus a weighted combination of many shallow trees, enabling the approximation of highly complex data relationships [Friedman, 2001].
Once we choose a parameterized model $F(\mathbf{x}; \mathbf{P})$, the functional optimization problem can

[2]Later on, we will also discuss alternative base learners.

be reformulated as an optimization problem over the parameters $\mathbf{P}$. Specifically, we look for the parameter vector $\mathbf{P}^*$ that minimizes the expected loss:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \Phi(\mathbf{P}),$$

where

$$\Phi(\mathbf{P}) = \mathbb{E}_{y,\mathbf{x}}\big[L(y, F(\mathbf{x}; \mathbf{P}))\big].$$

Once the optimal parameters $\mathbf{P}^*$ are found, the optimal function is given by

$$F^*(\mathbf{x}) = F(\mathbf{x}; \mathbf{P}^*).$$

For most choices of $F(\mathbf{x}; \mathbf{P})$ and loss functions L, solving this problem requires numerical optimization methods. A common way to express the optimization process is

$$\mathbf{P}^* = \sum_{m=0}^{M} \mathbf{p}_m,$$

where $\mathbf{p}_0$ is the initial guess for the parameters, and $\{\mathbf{p}_m\}_{m=1}^{M}$ are successive updates (sometimes referred to as *steps* or *boosts*). Each update $\mathbf{p}_m$ depends on the sequence of all previous updates and is determined by the specific optimization algorithm being used [Friedman, 2001].
In practice, models such as XGBoost combine a parameterized representation of the predictor function (e.g., decision trees as base learners) with numerical optimization techniques (e.g., gradient boosting with second-order updates) to estimate the parameters, where the optimization is carried out iteratively through gradient-based updates.

Now we look the steepest-descent algorithm (on parameter space) is one of the most basic and widely used methods for numerical minimization [Friedman, 2001].
The following discussion of steepest descent in parameter space is provided for illustrative purposes; in the next section, we turn to steepest descent in function space, which constitutes the core optimization principle underlying XGBoost.
At each iteration m, the current gradient vector is computed as [Friedman, 2001]:

$$\mathbf{g}_m = \left\{ \left. \frac{\partial \Phi(\mathbf{P})}{\partial P_j} \right|_{\mathbf{P}=\mathbf{P}_{m-1}} \right\},$$

which represents the local rate of change of the objective function with respect to the parameters. The parameter vector at the previous step is [Friedman, 2001]

$$\mathbf{P}_{m-1} = \sum_{i=0}^{m-1} \mathbf{p}_i,$$

where $\mathbf{p}_i$ are the past update steps. The new update step is then chosen as [Friedman, 2001]

$$\mathbf{p}_m = -\rho_m \mathbf{g}_m,$$

meaning that we move in the direction of the negative gradient, which corresponds to the steepest decrease in the objective function. The step length ρ_m is determined by a line search [Friedman, 2001].:

$$\rho_m = \arg\min_{\rho} \Phi(\mathbf{P}_{m-1} - \rho \mathbf{g}_m).$$

Thus, the algorithm iteratively updates the parameters by moving in the direction of the steepest descent, with the step size chosen to minimize the objective function along that direction [Friedman, 2001].
We now focus on Steepest Descent in Function Space, the key optimization principle underlying XGBoost and many other gradient boosting algorithms. The objective is to determine the optimal function $F(\mathbf{x})$ that minimizes the expected loss [Friedman, 2001].:

$$\Phi(F) = \mathbb{E}_{y,\mathbf{x}}\big[L(y, F(\mathbf{x}))\big].$$

Unlike parameter optimization, here the object to optimize is the entire function $F(\mathbf{x})$.
We now present the incremental construction of $F(\mathbf{x})$: rather than solving directly for the entire function, we build it step by step,

$$F^*(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x}),$$

where:

- $f_0(\mathbf{x})$ is an initial guess (for example, a constant such as the mean of y),
- each $f_m(\mathbf{x})_1^M$ is an increment (a "boost") added at step m.

This is the principle of boosting: constructing F by sequentially adding weak learners. The following equation represents the steepest descent in function space. At iteration m, the update is chosen in the direction of the negative functional gradient [Friedman, 2001].:

$$f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x}),$$

where:

- $g_m(\mathbf{x})$ is the functional gradient of the loss with respect to F,
- ρ_m is a step size determined by line search.

The functional gradient is defined as:

$$g_m(\mathbf{x}) = \left[\frac{\partial \phi(F(\mathbf{x}))}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left[\frac{\partial\, \mathbb{E}_y\big[L(y, F(\mathbf{x})) \mid \mathbf{x}\big]}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}.$$

which means: take the derivative of the expected loss with respect to the function values, and evaluate it at the current approximation F_{m-1}. Thus, $g_m(\mathbf{x})$ is essentially the expected gradient of the loss at point $\mathbf{x}$ [Friedman, 2001].
Before adding the new increment, the current model is:

$$F_{m-1}(\mathbf{x}) = \sum_{i=0}^{m-1} f_i(\mathbf{x}).$$

Under mild regularity conditions, it is possible to interchange differentiation and expectation. With this assumption, the functional gradient can be rewritten as [Friedman, 2001]:

$$g_m(\mathbf{x}) = \mathbb{E}_y \left[\frac{\partial L(y, F(\mathbf{x}))}{\partial F(\mathbf{x})} \,\middle|\, \mathbf{x} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}.$$

The scalar multiplier ρ_m in equation $f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x})$, is determined through a line search. This means that we optimize the step size taken in the direction of the negative gradient so as to minimize the loss. Formally, it is defined as

$$\rho_m = \arg\min_{\rho} \mathbb{E}_{y,\mathbf{x}} \Big[L\big(y, F_{m-1}(\mathbf{x}) - \rho g_m(\mathbf{x})\big) \Big].$$

In other words, while $g_m(\mathbf{x})$ specifies the steepest-descent direction in function space, the parameter ρ_m controls how far we move along that direction. Choosing ρ_m in this way ensures that the update $f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x})$ leads to the largest possible reduction in the expected loss at iteration m [Friedman, 2001].

To recap, at each step, we move F in the direction of the steepest descent of the loss (the negative gradient). The increment f_m can be seen as "fitting to the residuals," while ρ_m determines how far to move in that direction [Friedman, 2001].

In practice, the nonparametric estimation of the conditional expectation $\mathbb{E}[y \mid \mathbf{x}]$ breaks down when only a finite data sample $\{y_i, \mathbf{x}_i\}_{i=1}^N$ is available. In such a case, $\mathbb{E}_y[\cdot \mid \mathbf{x}]$ [3] cannot be estimated accurately at each $\mathbf{x}_i$, and even if it could, one would still prefer to obtain an estimate $F^*(\mathbf{x})$ that generalizes to values of $\mathbf{x}$ not present in the training sample. To achieve this, one borrows strength from nearby data points by imposing smoothness on the solution. A common way to introduce such structure is to assume a parameterized functional form, as the one that we have seen before $F(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M) = \sum_{m=1}^M \beta_m\, h(\mathbf{x}; \mathbf{a}_m)$, where $h(\mathbf{x}; \mathbf{a}_m)$ are basis functions (e.g., regression trees), $\mathbf{a}_m$ are their internal parameters, and β_m are the associated coefficients (weight). The problem then reduces to parameter optimization, namely finding the values $\{\beta_m, \mathbf{a}_m\}_1^M$ that minimize the empirical loss [Friedman, 2001]:

$$\{\beta_m, \mathbf{a}_m\}_1^M = \arg\min_{\{\beta_m, \mathbf{a}_m\}_1^M} \sum_{i=1}^N L\left(y_i, \sum_{m=1}^M \beta_m\, h(\mathbf{x}_i; \mathbf{a}_m) \right).$$

In real-world applications, directly solving for all parameters $\{\beta_m, \mathbf{a}_m\}_1^M$ in a single optimization problem is often computationally infeasible. A common alternative is to adopt a *greedy stagewise approach*, where the model is built sequentially by adding one basis function at a time [Friedman, 2001]. Specifically, at each iteration $m = 1, 2, \ldots, M$, the parameters of the new basis function are obtained by solving

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^N L\big(y_i, F_{m-1}(\mathbf{x}_i) + \beta\, h(\mathbf{x}_i; \mathbf{a})\big), \tag{1}$$

where F_{m-1} is the model constructed up to step $m-1$. The model is then updated by adding the newly fitted basis function [Friedman, 2001]:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m\, h(\mathbf{x}; \mathbf{a}_m).$$

[3] The notation $\mathbb{E}[y \mid \mathbf{x}]$ denotes the conditional mean of y. By contrast, $\mathbb{E}_y[\cdot \mid \mathbf{x}]$ represents the expectation operator applied to a generic function of y, where the dot "$\cdot$" serves as a placeholder.

This procedure incrementally improves the model by greedily selecting the function $h(\mathbf{x}; \mathbf{a}_m)$ and its coefficient β_m that most reduces the empirical loss at each step. It is important to note that the stagewise strategy differs from stepwise approaches, as previously added terms are not readjusted when new ones are introduced. In machine learning, the stagewise procedure defined in the last two equations is referred to as boosting. The individual functions $h(\mathbf{x}; \mathbf{a})$ are called weak learners or base learners, and in most applications they are implemented as classification or regression trees [Friedman, 2001].
In cases where solving the optimization problem in (4.2) for a given loss function $L(y, F)$ and base learner $h(\mathbf{x}; \mathbf{a})$ is computationally intractable or analytically difficult to obtain, the solution can be approximated by interpreting the update step as a steepest descent step in function space [Friedman, 2001].
Formally, given the current model $F_{m-1}(\mathbf{x})$, the negative gradient of the loss with respect to the model prediction at each data point is computed as

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(x)=F_{m-1}(\mathbf{x})}.$$

These negative gradients, often referred to as *pseudo-residuals*, represent the direction in which the loss decreases most rapidly (best greedy). However, they are only defined at the observed data points $\{\mathbf{x}_i\}_{i=1}^N$ and cannot be used directly as an unconstrained functional update [Friedman, 2001].
To address this limitation, the idea is to approximate the pseudo-residuals by fitting a function from the chosen class of base learners $h(\mathbf{x}; \mathbf{a})$. Specifically, one selects the function $h(\mathbf{x}; \mathbf{a}_m)$ that is most highly correlated with the negative gradient by solving

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} \left[-g_m(\mathbf{x}_j) - \beta h(\mathbf{x}_j; \mathbf{a})\right]^2.$$

This least-squares fitting step provides the best approximation to the gradient direction within the restricted family of base learners [Friedman, 2001].
Once the base learner $h(\mathbf{x}; \mathbf{a}_m)$ is selected, the optimal step size ρ_m is determined by a one-dimensional line search:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L\Big(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)\Big).$$

The model is then updated as

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m).$$

In essence, this procedure replaces the original difficult optimization problem (4.2) with a two-step approach: (i) fitting the weak learner to the pseudo-residuals via least-squares, and (ii) performing a simple one-dimensional optimization to determine the step size. This makes the algorithm applicable to *any differentiable loss function* $L(y, F)$, while still following the forward stagewise additive modeling framework [Friedman, 2001].

Algorithm 1 Gradient Boost (Generic Workflow)

1: **Input:** Training data $\{(x_i, y_i)\}_{i=1}^{N}$, loss function $L(y, F)$, number of iterations M.
2: **Initialize:**

$$F_0(x) = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, \rho)$$

3: **for** $m = 1$ to M **do**
4: Compute pseudo-residuals:

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}(x)}, \quad i = 1, \ldots, N$$

5: Fit base learner $h(x; a_m)$ to pseudo-residuals by solving:

$$a_m = \arg\min_{a,\beta} \sum_{i=1}^{N} \big(\tilde{y}_i - \beta h(x_i; a)\big)^2$$

6: Compute optimal step size:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L\big(y_i, F_{m-1}(x_i) + \rho h(x_i; a_m)\big)$$

7: Update model:

$$F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m)$$

8: **end for**
9: **Output:** Final model $F_M(x)$ [Friedman, 2001].

In practice, least-squares fitting is commonly used to approximate the negative gradient, as it provides a computationally efficient and convenient choice among possible fitting criteria [Friedman, 2001].

k-fold cross validation
As discussed in the chapter on Multiple, Ridge, and Lasso regression, k-fold cross-validation is applied to the training set in order to identify the optimal parameters. In the case of Ridge and Lasso, this procedure was used to determine the optimal shrinkage parameter, whereas for XGBoost it is employed both to select the most suitable base learner and to tune the relevant hyperparameters (as illustrated in the code provided at the end of the chapter).

Alternative base learners
During the implementation, I experimented with different choices of base learners.
Decision trees - as previously mentioned - are generally employed as the default base learners in XGBoost, as boosted tree ensembles have demonstrated consistently strong predictive performance across a wide range of applications. Nevertheless, alternative base learners are also available in XGBoost. For instance, `gblinear` implements a gradient-boosted

linear model, while `DART` extends decision tree boosting by incorporating a dropout mechanism inspired by neural networks. In addition, XGBoost provides an implementation of random forests as another possible option [Wade, 2020]. In the following, we provide an overview of the available base learners in XGBoost, without delving into their mathematical formulation.
In contrast to decision trees, which are particularly effective for modeling non-linear relationships, linear models may be more appropriate when the underlying data structure is linear. For such cases, XGBoost provides the option of using `gblinear` as a base learner. The principle of boosted linear models follows the same framework as boosted trees: each subsequent model is trained on the residuals of the previous one, and the ensemble prediction is obtained by summing the outputs. The distinction lies in the fact that, with `gblinear`, all models in the ensemble are linear. Moreover, `gblinear` incorporates regularization terms similar to those used in Lasso and Ridge regression, which enhance generalization and control overfitting. Depending on the configuration, repeated boosting with `gblinear` can approximate a single Lasso regression [Wade, 2020].
DART (Dropouts meet Multiple Additive Regression Trees) was introduced as a variation of boosted trees to mitigate the excessive dependency on previous models that characterizes Multiple Additive Regression Trees (MART). Instead of relying solely on shrinkage as a penalization mechanism, DART incorporates the dropout technique, inspired by neural networks, which removes nodes from each layer of learning. This strategy slows down the learning process and reduces overfitting by eliminating part of the information from earlier rounds. In each boosting iteration, rather than summing the residuals of all previous trees, DART selects a random subset of trees and rescales their contributions by a factor of $1/k$, where k denotes the number of dropped trees. The XGBoost implementation of DART builds upon `gbtree` and introduces additional hyperparameters to handle the dropout mechanism, thereby extending the flexibility of the model while preserving the interpretability of tree-based learners [Wade, 2020].
XGBoost also allows the use of random forests as base learners, which can be implemented. While gradient boosting was originally designed to enhance the performance of weak base learners rather than strong learners such as random forests, there may still be cases where random forests provide advantages. The XGBoost implementation of random forests differs from traditional approaches by including default hyperparameters to mitigate overfitting and customized methods for constructing trees [Wade, 2020].

tuning hyperparameters

Hyperparameter tuning can be performed either by exploring the entire parameter space simultaneously or by optimizing each parameter individually [Wade, 2020]. In my implementation, I first analyzed the impact of each hyperparameter separately and then combined them to refine the overall optimization process. We now turn our attention to the hyperparameters that were tuned during the implementation process.

- $n_{\text{estimators}}$: number of trees in the ensemble. In XGBoost, this corresponds to the number of boosting rounds, i.e., trees fitted sequentially on the residuals.

- learning_rate: controls the contribution of each tree by scaling its weights. Lower values reduce overfitting but require more trees to achieve good performance.

- max_depth: maximum depth of individual trees. Deeper trees capture more complex relationships but risk overfitting.

- Gamma (γ): minimum loss reduction required to make a further split. Acts as a regularization parameter, where higher values result in more conservative models.

- subsample: fraction of training instances (rows) randomly sampled for each boosting round. Values below 1.0 reduce variance and help prevent overfitting.

- colsample_bytree: fraction of features (columns) randomly sampled when constructing each tree. Helps reduce correlation between trees and lowers variance.

- colsample_bylevel / colsample_bynode: refinements of feature subsampling—bylevel samples columns for each tree depth, and bynode for each split.

[Wade, 2020].
For readers interested in a more detailed description of the hyperparameters, the official XGBoost documentation is available at `https://xgboost.readthedocs.io/en/latest/parameter.html`.

Ensemble Learning: Voting and Stacking Approaches

As the final step of the implementation, we introduce two ensemble methods—Voting Classifier and Stacking—which are employed to combine the strengths of multiple models in order to enhance predictive accuracy and robustness.
The VotingClassifier combines multiple classification models by applying majority voting, while the VotingRegressor averages predictions from multiple regression models [Wade, 2020].
Stacking is an ensemble learning technique that integrates models at two levels. At the base level, multiple models generate predictions from the original data. At the meta level, these predictions are used as inputs for a meta-model, which produces the final output. Unlike standard ensembles, stacking does not rely directly on the raw data at the final stage, but rather on the predictions of base learners. This hierarchical structure often enhances predictive performance, especially when diverse models are combined, and it is common practice to adopt simple meta-models, such as linear or logistic regression, to avoid unnecessary complexity [Wade, 2020].
Before employing ensemble methods such as the VotingClassifier or stacking approaches, it is crucial to compute a correlation matrix of the candidate models' predictions. Highly correlated models contribute redundant information and may reduce the overall effectiveness of the ensemble. Therefore, removing or replacing models with strong correlations helps improve diversity among base learners, which in turn enhances the robustness and generalization ability of the ensemble [Wade, 2020].

Practical Implementation of XGBoost model

The first step in the XGBoost implementation consisted of eliminating highly correlated features, although this procedure is not strictly necessary for predictive performance. Nevertheless, it was applied to enhance computational efficiency, improve interpretability,

mitigate overfitting, and reduce overall model complexity. Feature selection was conducted through correlation analysis and mutual information.
During the model fitting phase, both cross-validation and K-fold validation were employed to conduct the analysis. At the fitting stage, the evaluation relied primarily on the Coefficient of Determination (R^2) and Mean Squared Error (MSE). Subsequently, for the validation and test phases, the full set of evaluation criteria outlined in the previous section was applied in order to ensure a comprehensive assessment of predictive performance.
The base learner selected for the XGBoost implementation was `gblinear`, as it yielded the best performance during the evaluation of alternative base learners.
In conclusion, the best evaluation metrics were obtained with the Stacking Regressor, which outperformed the hyperparameter-tuned XGBoost model.
The complete code implementation and results are available in the GitHub repository: XGboost.ipynb.

Bibliography

Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

Syed Mohtashim Mian, Mohammad Shuaib Khan, Mohd Shawez, and Amandeep Kaur. Artificial intelligence (ai), machine learning (ml) & deep learning (dl): A comprehensive overview on techniques, applications and research directions. In *2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, pages 1404–1409. IEEE, IEEE, 2024. ISBN 979-8-3503-7999-0. doi: 10 . 1109 / ICSCSS60660 . 2024 . 10625198.

Antonio Rafael Sabino Parmezan, Vinicius M. A. Souza, and Gustavo E. A. P. A. Batista. Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model. *Information Sciences*, 484:302–337, 2019. doi: 10.1016/j.ins.2019.01.076.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Limited, Harlow, United Kingdom, 4th edition, 2021. ISBN 978-1-292-40111-3. Global Edition.

Jo Taeho. *Deep Learning Foundations*. Springer, 2023.

Corey Wade. *Hands-On Gradient Boosting with XGBoost and scikit-learn: Perform accessible machine learning and extreme gradient boosting with Python*. Packt Publishing, Birmingham, UK, 2020.