

MachineLearnAthon - Microlecture

Topic XGBoost

Introduction to XGBoost Algorithm

MachineLearnAthon
A project Co-funded by the Erasmus+ programme of the European Union

15.10.2024



Overview of XGBoost and Boosting

- XGBoost is a scalable and accurate machine learning library for gradient boosting.
- Boosting is an ensemble technique where weak models (decision trees) are built sequentially.
- Minimizes loss by adding trees based on gradient descent on residuals.

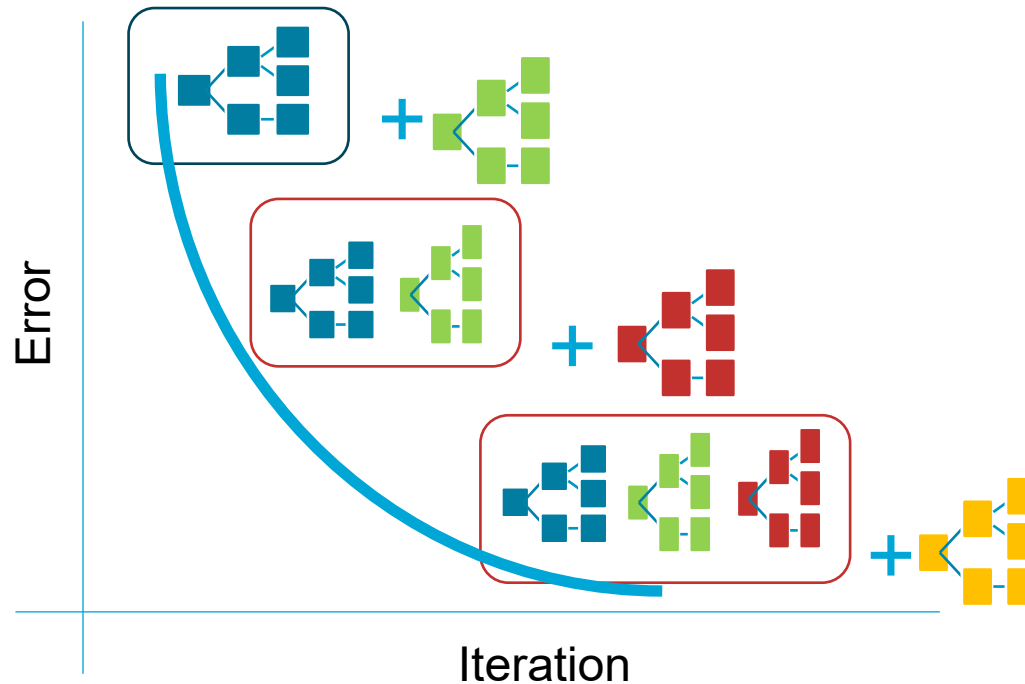


$\text{Residuals1} > \text{Residuals 2} > \text{Residuals 3} > \dots > \text{Residuals n}$



Decision Trees as Base Learners

- Decision Trees split data based on feature values to make predictions.
- XGBoost combines multiple weak learners (simple decision trees).
- Base Learners are shallow trees, designed to learn from residual errors.
- Each tree is built to correct the errors of the previous tree.



Regularization, Overfitting, and Learning Rate

- **Overfitting** occurs when a model learns the noise in the training data, reducing generalization.
- Regularization controls overfitting by adding penalties for complexity.
- L1 Regularization (Lasso) and L2 Regularization (Ridge) penalize large model weights.
- Lambda (λ) is the regularization parameter in XGBoost.
- Learning Rate (η) controls the contribution of each tree.
- Lower learning rates improve accuracy but require more trees.

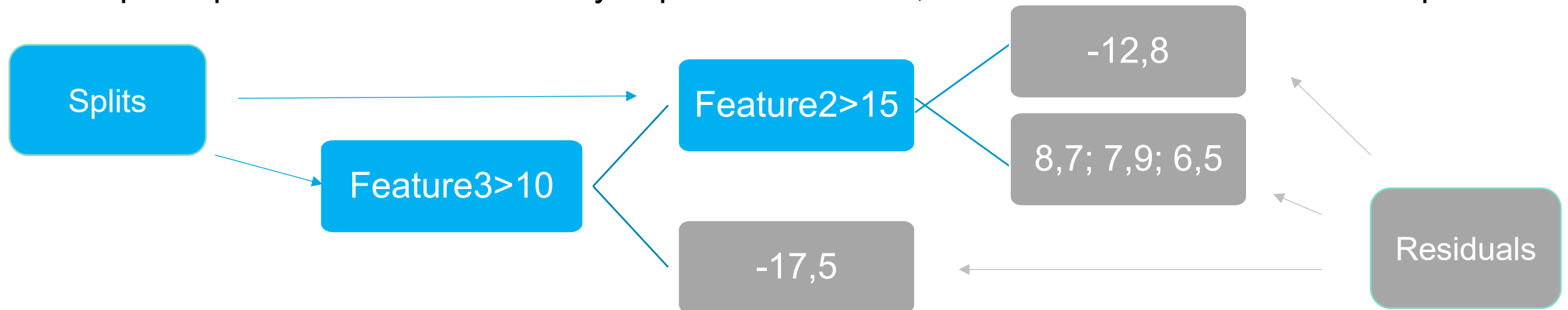
Minimizing Residuals with Gradient Descent

- XGBoost uses gradient descent to minimize residual errors.
- Each tree corrects residual errors from previous iterations.
- Iteratively builds trees until the overall loss is minimized.
- **Output formula:** $\hat{y}_{\text{final}} = \hat{y}_1 + \eta * \hat{y}_2 + \eta * \hat{y}_3 + \dots + \eta * \hat{y}_n$
 - η is the learning rate
 - \hat{y}_n is the prediction of the n-th tree, corrected iteratively.



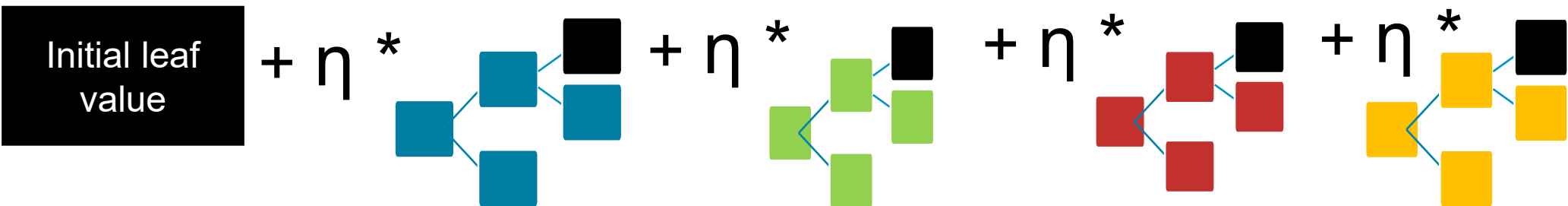
Starting with a Single Leaf Node

- Step 1: XGBoost begins with a single leaf node containing the mean target value.
- Step 2: Splits are computed by evaluating various feature thresholds.
- Step 3: XGBoost calculates **the similarity score** for each split to assess its quality.
- Step 4: Splits are made when they improve the model, otherwise the leaf remains unsplit.



Calculating Splits and Growing the Tree

- Step 1: After splitting, data is divided into branches.
- Step 2: Gain is calculated to decide the best split. Higher gain = better split.
- Step 3: The tree grows until a stopping criterion (max depth or min gain) is met.
- Step 4: Pruning stops the tree growth if no further splits improve the model.
- Learning Rate (η) scales the contribution of each tree.

$$\text{Prediction} = \text{Initial leaf value} + \eta * \text{Tree 1} + \eta * \text{Tree 2} + \eta * \text{Tree 3} + \eta * \text{Tree 4}$$




Key XGBoost Concepts

- **Similarity Score:** Measures how homogeneous a node is. Higher score = better prediction.

$$\text{Similarity} = (\text{sum of residuals})^2 / (\text{number of residuals} + \lambda)$$

- **Gain:** Improvement from splitting a node. Higher gain = more significant split.

$$\text{Gain} = (\text{similarity left} + \text{similarity right} - \text{similarity of parent})$$

- **Pruning:** Stops tree growth when further splits do not improve the model.
- **Lambda (λ):** Regularization parameter that penalizes complexity to prevent overfitting.
- **Learning Rate (η):** Controls how much each tree contributes to the final prediction.



XGBoost Efficiency Features

- Sparsity-aware algorithms handle missing data efficiently.
- Block structure enables parallel computation, speeding up training.
- Out-of-core computing supports large datasets by loading them in chunks.
- Pruning strategies reduce computation by avoiding unnecessary branches.



XGBoost in Regression Problems

- **Used for predicting continuous target variables**, where the goal is to predict a real-valued number rather than a class.
- Minimizes regression loss functions like **Mean Squared Error (MSE)**.
- It can also minimize **Mean Absolute Error (MAE)** for regression tasks, depending on the objective set.
- The final prediction is the sum of all tree predictions, with each tree's contribution scaled by the learning rate.
- **Scalability**: XGBoost handles large datasets efficiently through parallel processing, and it can also deal with missing values natively.
- Example: Price prediction, demand forecasting.



XGBoost in Classification Problems

- Used for **categorical target variables** in classification problems (binary or multi-class).
- Minimizes **logistic loss function** (for binary classification).

$$\text{Log - Loss function} = \sum_{i=0}^n -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

Where: n is the number of samples, indexed by i , y_i is the true class for the index i , and p_i is the model prediction for the index i

- For **multi-class classification**, XGBoost uses **softmax**
 - *Assigns probabilities across multiple classes; the sum of probabilities equals 1*
- **Iteratively builds trees** to reduce misclassifications by focusing on residual errors.
- **Weighting:** Misclassified examples get higher weights in subsequent trees to focus on hard-to-classify cases.
- **Prediction:** The class with the highest probability is the final prediction.



Thank you very much for your attention!



Literature

- Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32.
<https://doi.org/10.1023/A:1010933404324>
- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). ACM.
<https://doi.org/10.1145/2939672.2939785>
- Chen, T., & He, T. (2015). xgboost: eXtreme Gradient Boosting (R package vignette). Retrieved from
<https://cran.r-project.org/web/packages/xgboost/index.html>
- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), 1189–1232. <https://doi.org/10.1214/aos/1013203451>
- GeeksforGeeks. (2023, February 6). XGBoost. <https://www.geeksforgeeks.org/xgboost/>
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer. <https://doi.org/10.1007/978-0-387-84858-7>

Literature (cont.)

- He, T., & Chen, T. (2015). Practical lessons from predicting clicks on ads at Facebook. KDD Workshop on Industrial Practice.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems, 30.
https://papers.nips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html
- Nielsen, D. (2016). Tree boosting with XGBoost: Why does XGBoost win "every" machine learning competition? arXiv preprint arXiv:1603.02754. <https://arxiv.org/abs/1603.02754>
- XGBoost Developers. (2024). XGBoost documentation. Retrieved from <https://xgboost.readthedocs.io/>

The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

This material is licenced under CC BY-NC-ND 4.0
(<https://creativecommons.org/licenses/by-nc-nd/4.0/>).