XGBoost

A Rameswar Patro

National Institute of Science Education and Research, Bhubaneswar

XGBoost

XGBoost, short for **Extreme Gradient Boosting,** has emerged as a leading machine learning model due to its exceptional performance across various tasks. Gradient boosting is a powerful ensemble technique that combines weak learners (e.g., decision trees) using the gradient descent architecture to enhance prediction accuracy progressively. The key advantage of gradient boosting is that it can iteratively improve the model's performance by focusing on areas where the previous models struggled. This approach can lead to highly accurate models, especially for complex problems. XGBoost is a refined implementation renowned for speed, scalability, and regularization capabilities.

Objective function

In XGBoost, the objective function quantifies how well the model is performing and guides the optimization algorithm in finding the best parameters to minimize or maximize this function.

The objective function in XGBoost consists of two main parts the **loss function** and the **regularization term**. It can be expressed as:

$$\text{Objective}(\Theta) = \text{Loss}(y, \hat{y}) + \text{Regularization}(\Theta),$$

where Θ represents the model parameters to be optimized (e.g., tree structure, leaf weights), y is the true output, and $\hat{y}$ is the predicted output.

We are able to minimize the objective function using optimization techniques like gradient descent.

Loss Function

We use different loss functions depending on the use case.

1. Mean Squared Error (MSE):
   Use Case: Regression
   Function: $\text{Loss}(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
2. Cross-Entropy Loss:
   Use Case: Binary Classification
   : $\text{Loss}(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$

Regulariation

There are two main types of regularization used in XGBoost:

1. **L1 Regularization (Lasso)**: Mathematically, it can be expressed as the sum of the absolute values of the model weights:
   $$\text{Regularization}(\Theta) = \frac{1}{2}\lambda\Sigma|\theta_i|$$
   where λ is the regularization parameter and θ_i's are the model parameters. L1 regularization encourages sparsity in the model by driving some of the model parameters to zero, effectively selecting a subset of features that are most informative for prediction.
2. **L2 Regularization (Ridge)**: Unlike L1 regularization, L2 regularization adds a penalty term proportional to the square of the model parameters:
   $$\text{Regularization}(\Theta) = \frac{1}{2}\lambda\Sigma\theta_i^2$$
   where λ is the regularization parameter and θ_i's are the model parameters. L2 regularization tends to shrink the parameters towards zero without encouraging sparsity, leading to smoother models with smaller parameter values.

XGBoost for Regression

In the first step XGBoost determines the structure of the tree greedily in each level. To find out the best possible structure of the tree, XGBoost uses the similarity score at each level defined by

$$\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma,$$

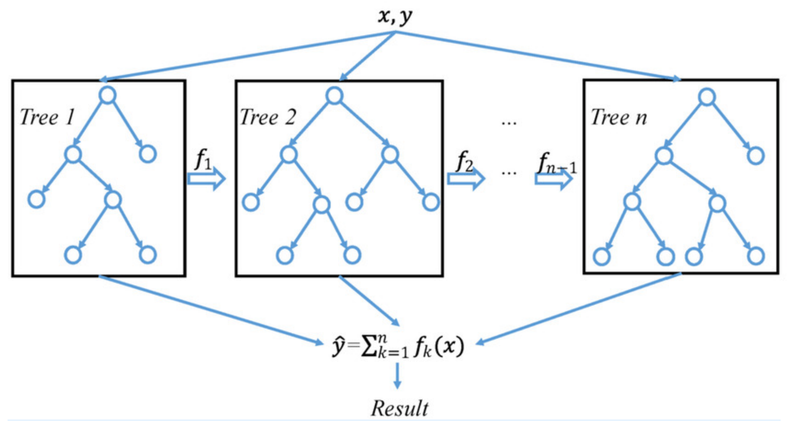
where G_L = Sum of the residuals in left child node, and H_L = Number of residuals in the left child node. The score depends on the specific threshold and the specific feature selected. Ideally the best theshlod and feature pair is selected at each level. Analytically solving the objective function, we get the out put value at each leaf as

$$\frac{\text{Sum of Residuals}}{\text{Number of residuals} + \lambda}.$$

Optimizations

XGBoost implements several optimizations to enhance efficiency, scalability, and performance. Here are some key optimizations in XGBoost:

- **Cache-aware Access**: XGBoost optimizes memory access patterns by utilizing cache-aware data structures and algorithms. This reduces memory latency and improves training speed by minimizing the number of cache misses.
- **Approximate Tree Learning**: XGBoost implements algorithms for approximate tree learning, such as histogram-based methods, to speed up tree construction. These methods enable efficient computation of split points and reduce the computational overhead of tree building.
- **Parallel and Distributed Computing**: XGBoost supports parallel and distributed computing, leveraging multi-core CPUs and distributed computing frameworks like Dask or Spark. It parallelizes tree construction and gradient computations to utilize hardware resources effectively and scale to large datasets.



Applications

XGBoost has found wide-ranging applications across various domains due to its exceptional performance, scalability, and versatility. Some notable applications include:

- Credit Scoring: XGBoost is used for credit risk assessment and determining the likelihood of default by analyzing customer data.
- Disease Diagnosis: XGBoost models assist in diagnosing diseases based on patient data, medical imaging, and genetic information.