Let's start with BOOSTING in ML:

For that we would first learn Ensemble Learning,

ENSEMBLE LEARNING: When you combine and take leverage of multiple models to get your task done.

BOOSTING: It is a type of ensemble learning where we train a dataset sequentially on multiple weak models so that they can together give a better output, we would also discuss further how it is done, like different boosting techniques do this task differently, so lets learn about some of the important boosting algorithms.

AdaBoost (Adaptive Boosting):

1.Focus on Misclassifications:
• AdaBoost focuses on improving the areas where previous models made errors.
• It adjusts the weights of incorrectly classified instances, increasing their importance for the next model.

2. Weight Update:
• After each weak learner is added, AdaBoost adjusts the weights of the data points.
• Misclassified points get higher weights, and correctly classified points get lower weights.

3. Model Combination:
• The final model is a weighted sum of all the weak learners, where each learner's weight depends on its accuracy.
• Weak learners are combined by assigning each a weight based on its error rate.

4. Algorithm:
•The AdaBoost algorithm is relatively simpler and typically uses decision stumps (single-level decision trees) as weak learners.

5.Error Minimization:
•AdaBoost works by reducing the classification error directly.

TAKEAWAY: In AdaBoost, we generally do increase the weights of false predictions so that the mode next in the sequence could focus more on that wrongly predicted values and try to rectify it.

Gradient Boosting.

In gradient boosting, each new model tries to correct the errors (residuals) made by the previous model.

Step 1: Initial Prediction

- **Start with an initial model:** This is often a simple model, like the mean of the target values for regression problems.
- For example, if our target values Y are [1.5, 1.7, 3.2, 4.0, 5.1], the initial prediction could be the mean of these values, which is approximately 3.1

Step 2: Calculate Residuals

- **Compute residuals:** Residuals are the differences between the actual values and the predictions made by the current model.
- For each data point i, the residual is calculated as:

$$\text{Residual}_i = \text{Actual Value}_i - \text{Predicted Value}_i$$

- For our example, if our initial prediction is 3.1, the residuals would be:

$$\text{Residual}_1 = 1.5 - 3.1 = -1.6$$
$$\text{Residual}_2 = 1.7 - 3.1 = -1.4$$
$$\text{Residual}_3 = 3.2 - 3.1 = 0.1$$
$$\text{Residual}_4 = 4.0 - 3.1 = 0.9$$
$$\text{Residual}_5 = 5.1 - 3.1 = 2.0$$

Step 3: Fit a Model to Residuals

- **Train a new model to predict the residuals:** The new model (often a decision tree) is trained to learn the patterns in the residuals.
- This model tries to capture what the initial model missed.

Step 4: Update the Overall Prediction

- **Update the overall prediction:** Add the predictions of the new model to the current predictions. This is usually done with a learning rate to control how much the new model's predictions influence the overall model.

- For example, if the new model predicts the residuals as accurately as $[-1.6, -1.4, 0.1, 0.9, 2.0]$ and we use a learning rate of 0.1, the updated predictions would be:

$$\text{Updated Prediction}_1 = 3.1 + 0.1 \times (-1.6) = 3.1 - 0.16 = 2.94$$
$$\text{Updated Prediction}_2 = 3.1 + 0.1 \times (-1.4) = 3.1 - 0.14 = 2.96$$
$$\text{Updated Prediction}_3 = 3.1 + 0.1 \times 0.1 = 3.1 + 0.01 = 3.11$$
$$\text{Updated Prediction}_4 = 3.1 + 0.1 \times 0.9 = 3.1 + 0.09 = 3.19$$
$$\text{Updated Prediction}_5 = 3.1 + 0.1 \times 2.0 = 3.1 + 0.2 = 3.3$$

Step 5: Repeat

- **Iterate the process:** Repeat steps 2 to 4 for a specified number of iterations or until the residuals are sufficiently small.
  - In each iteration, calculate new residuals based on the updated predictions.
  - Fit a new model to these new residuals.
  - Update the overall predictions by adding the scaled predictions of the new model.

Example of Iteration:

Second Iteration:

1. **Calculate New Residuals:**
   - Using the updated predictions from the first iteration, calculate the new residuals.

$$\text{New Residual}_1 = 1.5 - 2.94 = -1.44$$
$$\text{New Residual}_2 = 1.7 - 2.96 = -1.26$$
$$\text{New Residual}_3 = 3.2 - 3.11 = 0.09$$
$$\text{New Residual}_4 = 4.0 - 3.19 = 0.81$$
$$\text{New Residual}_5 = 5.1 - 3.3 = 1.8$$

2. **Fit New Model to New Residuals:**
   - Train another model to predict these new residuals.
3. **Update Predictions:**
   - Add the predictions of the new model (scaled by the learning rate) to the updated predictions from the first iteration.

XGBoost (Extreme Gradient Boosting):It is more advanced version of Gradient boosting that tried to work on the shortcomings of Gradient Boosting.

Overview

- **Enhanced Version**: XGBoost is an implementation of gradient boosting with several enhancements designed to improve performance and efficiency. It is widely used in machine learning competitions and real-world applications due to its speed and accuracy.
- **Algorithm**: Similar to gradient boosting but includes additional regularization and optimization techniques.

Key Features

- **Regularization**: Includes L1 (lasso) and L2 (ridge) regularization to prevent overfitting, making the model more generalizable.
- **Sparsity Awareness**: Handles sparse data efficiently by automatically treating missing values and optimizing for sparsity.
- **Weighted Quantile Sketch**: Uses an advanced method to handle weighted data points, improving the accuracy of split finding in decision trees.
- **Parallel Processing**: Supports parallel computation to speed up training, making it faster on large datasets.
- **Tree Pruning**: Implements a sophisticated tree pruning technique called "max depth" to avoid unnecessary splits and reduce overfitting.
- **Cross-Validation**: Includes built-in cross-validation capabilities to evaluate model performance during training.

Additional Features in XGBoost

- **Regularization**: Adds regularization terms to the objective function to penalize complex models.
- **Handling Missing Values**: Automatically handles missing data.
- **Column Block for Parallel Learning**: Optimizes memory usage and computation efficiency.
- **Cache Awareness**: Optimizes for better cache usage in CPUs.

Pros and Cons

- **Pros**:
  - Much faster and more efficient than standard gradient boosting.
  - Incorporates regularization to reduce overfitting.
  - Handles large datasets and high-dimensional data well.
  - Built-in support for handling missing values.
- **Cons**:
  - More complex and harder to tune due to additional parameters.
  - Can still be prone to overfitting if not properly regularized.

Feature	Gradient Boosting	XGBoost
Regularization	Basic	L1 and L2 regularization
Efficiency	Slower	Highly optimized, faster
Parallel Processing	Limited	Extensive support for parallel computation
Handling Missing Values	Basic	Advanced, automatic handling
Tree Pruning	Basic	Sophisticated pruning techniques
Memory Usage	Higher	Optimized for lower memory usage
Model Tuning	Easier, fewer parameters	More complex, more parameters
Performance	Good	Excellent, especially for large datasets