Boosting

Boosting refers to any kind of 'ensemble' learning in machine learning. Ensemble learning combines many weak learners to create a strong learner, in this, models are trained sequentially, and the models build on the trained predictors of its previous models.

There are a few types of boosting – AdaBoost, Gradient Boost, XGBoost

XGBoost

XGBoost short for Extreme Gradient Boosting is an optimized implementation of Gradient Boosting.

- Decision trees are used as its base learners, they are trained sequentially and each tree corrects the errors of the previous one.
- It has built in parallel processing to train models on large datasets quickly.

How it works?

1. Start with a base learner: The first decision tree is trained on the data. This simply predicts the average of the target variable when dealing with regression.
2. Calculate the errors: The errors between the predicted and actual values are calculated.
3. Train the next tree: The next tree is trained on the errors of the previous tree. This is done to correct the errors made by the first tree.
4. Repeat: These steps are repeated until a stopping criterion is met.
5. Combine the predictions: The final prediction is the sum of the predictions from all trees.

Math behind XGBoost

As we read above, XGBoost is an iterative process. We start with an initial prediction set to 0. Moving forward, to predict, we add each tree to reduce errors. Mathematically, the model can be represented as:

$$\widehat{y_i} = \sum_{k=1}^{K} f_k\,(x_i)$$

Where,

$\widehat{y_i}$ is the final predicted value for the ith data point

K is the number of trees in the ensemble

$\mathrm{f_k}(x_i)$ represents the prediction of the kth tree for the ith data point

The objective function in XGBoost consists of two parts: a **loss function** and a **regularization term**. The loss function measures how well the model fits the data and the regularization term simplify complex trees. The general form of the loss function is:

$$obj(\theta) = \sum_{i}^{n} l(y_i\hat{y_i}) + \sum_{1}^{K} \Omega(f_k)$$

Where,

$l(y_i\hat{y_i})$ is the loss function which computes the difference between the true value y_i and the predicted value $\hat{y_i}$

$\Omega(f_k)$ is the regularization term which discourages overly complex terms

Now, instead of fitting the model all at once we optimize the model iteratively. We start with an initial prediction $\hat{y_i}^{(0)} = 0$ and at each step we add a new tree to improve the model. The updated predictions after adding the tth tree can be written as:

$$\hat{y_i}^{(t)} = \hat{y_i}^{(t-1)} + f_k(x_i)$$

Where,

$\hat{y_i}^{(t-1)}$ is the prediction from the previous iteration

$f_k(x_i)$ is the prediction of the tth tree for the ith data point

Finally, when deciding how to split the nodes in the tree we compute the **information gain** for every possible split. By calculating the information gain for every possible split at each node XGBoost selects the split that results in the largest gain which effectively reduces the errors and improves the model's performance.

What Makes XGBoost "eXtreme"?

XGBoost extends traditional gradient boosting by including regularization elements in the objective function, XGBoost improves generalization and prevents overfitting.

Advantages of XGBoost:

- **High Performance & Speed** – Optimized for fast execution and efficient memory usage.
- **Regularization** – Reduces overfitting compared to traditional gradient boosting.
- **Handles Missing Values** – Automatically learns the best way to handle them.

- **Parallel & Distributed Computing** – Speeds up training on large datasets.
- **Feature Importance** – Provides insights into which features are most important.
- **Works with Imbalanced Data** – Can handle skewed datasets effectively with custom loss functions.

Disadvantages of XGBoost:

- **Sensitive to Hyperparameters** – Requires careful tuning for best results.
- **Computationally Expensive** – Can be resource-intensive, especially with large datasets.
- **Not Ideal for Small Datasets** – May overfit when data is limited.
- **Hard to Interpret** – More complex than simpler models like decision trees.