**Boosting Theory**

## **What is Boosting?**

Boosting is an ensemble learning technique that combines many **weak learners** (usually decision trees) to create a strong learner.

* A weak learner is a simple model that performs slightly better than random guessing (e.g., a shallow decision tree).
* Boosting builds models **sequentially**, where each new model tries to fix the mistakes of the previous ones.

## **Key Idea**

1. Start with a simple model (weak learner).
2. Identify the samples that were misclassified.
3. Give more weight to those hard-to-classify samples.
4. Train the next model focusing on these harder samples.
5. Combine all weak learners into one strong model.

**Difference vs Bagging**

* **Bagging** (Bootstrap Aggregating): trains models in **parallel**, averages results → reduces variance.
* **Boosting**: trains models **sequentially**, each improves on previous → reduces bias.

## **Popular Boosting Algorithms**

1. **AdaBoost (Adaptive Boosting)**
   1. Adjusts weights of samples: misclassified → higher weight.
   2. Final prediction = weighted vote of all learners.
2. **Gradient Boosting**
   1. Builds new models to predict the **residual errors** (difference between actual & predicted).
   2. Uses gradient descent to minimize loss.
3. **XGBoost (Extreme Gradient Boosting)**
   1. Optimized version of Gradient Boosting (faster, handles missing data, regularization).
4. **LightGBM**
   1. Gradient boosting using **leaf-wise tree growth** (faster, memory efficient).
5. **CatBoost**
   1. Gradient boosting optimized for **categorical features**.

**AdaBoost**

## **Core Idea**

* Build **multiple weak classifiers** (usually Decision Stumps = trees with depth=1).
* Each classifier focuses on the **samples misclassified** by the previous ones.
* Misclassified points get **higher weights**, so the next learner pays more attention to them.
* Final prediction = **weighted majority vote** (classification) or weighted sum (regression).

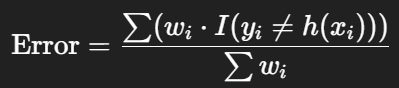
## **How AdaBoost Works (Algorithm)**

* **Initialize weights**

Each training sample gets equal weight

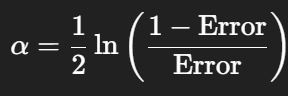


* **Train a weak learner** (e.g., decision stump).
* **Calculate error**



where I() is an indicator function.

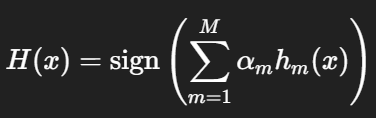
* **Compute model weight** (importance of learner)

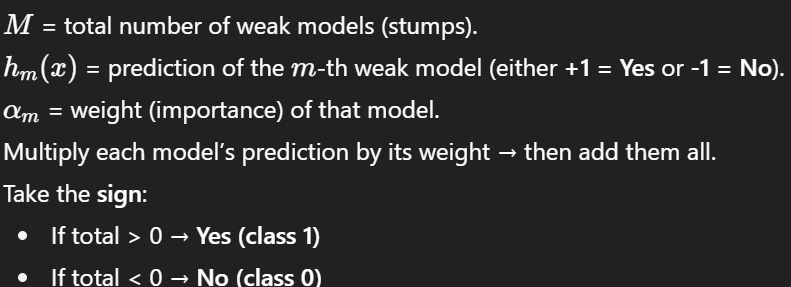


* **Update sample weights**
  + Misclassified samples → weight increases.
  + Correctly classified samples → weight decreases.



* **Normalize weights** so they sum to 1.
* **Repeat** steps 2–6 for *M* iterations.
* **Final Prediction**





## **Intuition**

* First classifier: weak, makes many mistakes.
* Second classifier: focuses on correcting those mistakes.
* Over time: ensemble becomes **much stronger**.

## **Pros**

* Works well for binary classification.
* Reduces bias significantly.
* Simple to implement.

## **Cons**

* Sensitive to noise and outliers.
* Can overfit if too many learners.

**Gradient Boosting**

## **What is Gradient Boosting?**

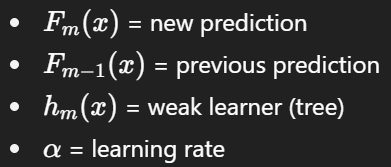
* Gradient Boosting is an ensemble method that builds models sequentially.
* Each new model tries to correct the errors made by the previous models.
* Uses decision trees (usually shallow ones) as weak learners.
* Instead of simple voting/averaging (like Bagging/Random Forests), it optimizes a loss function using gradient descent.

## **How it Works (Step-by-Step)**

1. Start with a base prediction (often the mean of target values for regression, or log odds for classification).
2. Compute the errors (residuals) between the prediction and actual values.
3. Fit a small decision tree to predict these residuals.
4. Update the prediction:



where:



1. Repeat until the error stops improving.

## **Key Concepts**

* **Weak Learner:** Usually, a decision tree with max depth 1 (stump) or 2.
* **Learning Rate (α):** Shrinks the contribution of each new tree (prevents overfitting).
* **Number of Trees (**n\_estimators**):** More trees → better fit, but risk of overfitting.
* **Loss Function:** Guides the optimization.
  + Regression: Mean Squared Error (MSE).
  + Classification: Log Loss / Deviance.

## **Advantages**

* Handles both regression & classification.
* Works well with structured/tabular data.
* Can handle different loss functions.
* Robust to outliers (if tuned properly).

## **Disadvantages**

* Slower training compared to Random Forest.
* Sensitive to hyperparameters (learning rate, depth, trees).
* Not great for extremely high-dimensional sparse data (like raw text).

## **Important Hyperparameters**

* n\_estimators: Number of boosting rounds (trees).
* learning\_rate: Step size shrinkage (default ~0.1).
* max\_depth: Depth of trees.
* subsample: Fraction of samples used per tree (helps reduce overfitting).
* min\_samples\_split / min\_samples\_leaf: Minimum samples to split/leaf.