AdaBoost (Adaptive Boosting)

- AdaBoost is an **ensemble learning method**, specifically a **boosting** algorithm.
- **Boosting:** Builds the ensemble *sequentially*. Each new model attempts to correct the errors made by the previous models.
- **Adaptive:** It adapts by giving more weight to data points that previous models misclassified, forcing subsequent models to focus on these "harder" examples. It also weights the contribution of each weak learner based on its accuracy.

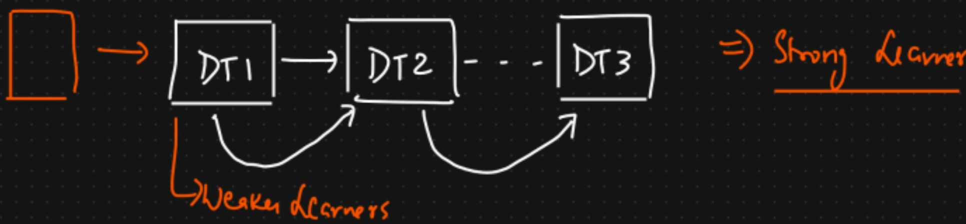
2. Core Idea & Intuition

Imagine you're studying for an exam with a group. Instead of everyone studying everything equally:

- First, everyone tries all topics (initial weak learner).
- You identify the topics where the group performed poorly (misclassified examples).
- The next study session focuses *more intensely* on these difficult topics (increasing weights for misclassified points).
- Someone who is particularly good at a certain topic gets their opinion weighted more heavily on questions related to that topic (weighting the weak learners).
- You repeat this process, adaptively focusing on weaknesses until the group (the ensemble) performs well overall.
- In this we make a decision tree of 1 level i.e. stumps.

AdaBoost does this with data points and simple models (weak learners).

Boosting {Sequentially connected}

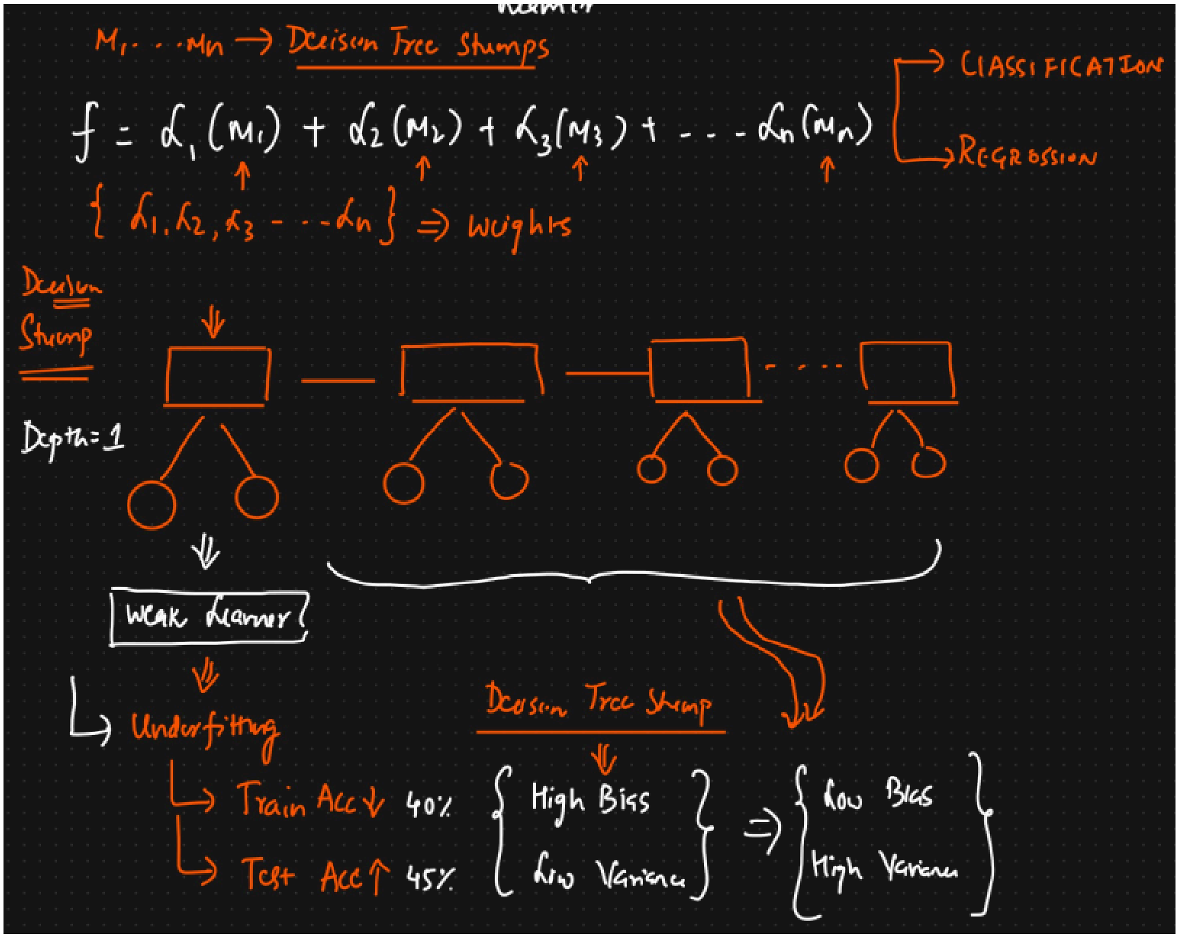


Weak Learner → Haven't learnt much from the Training Dataset

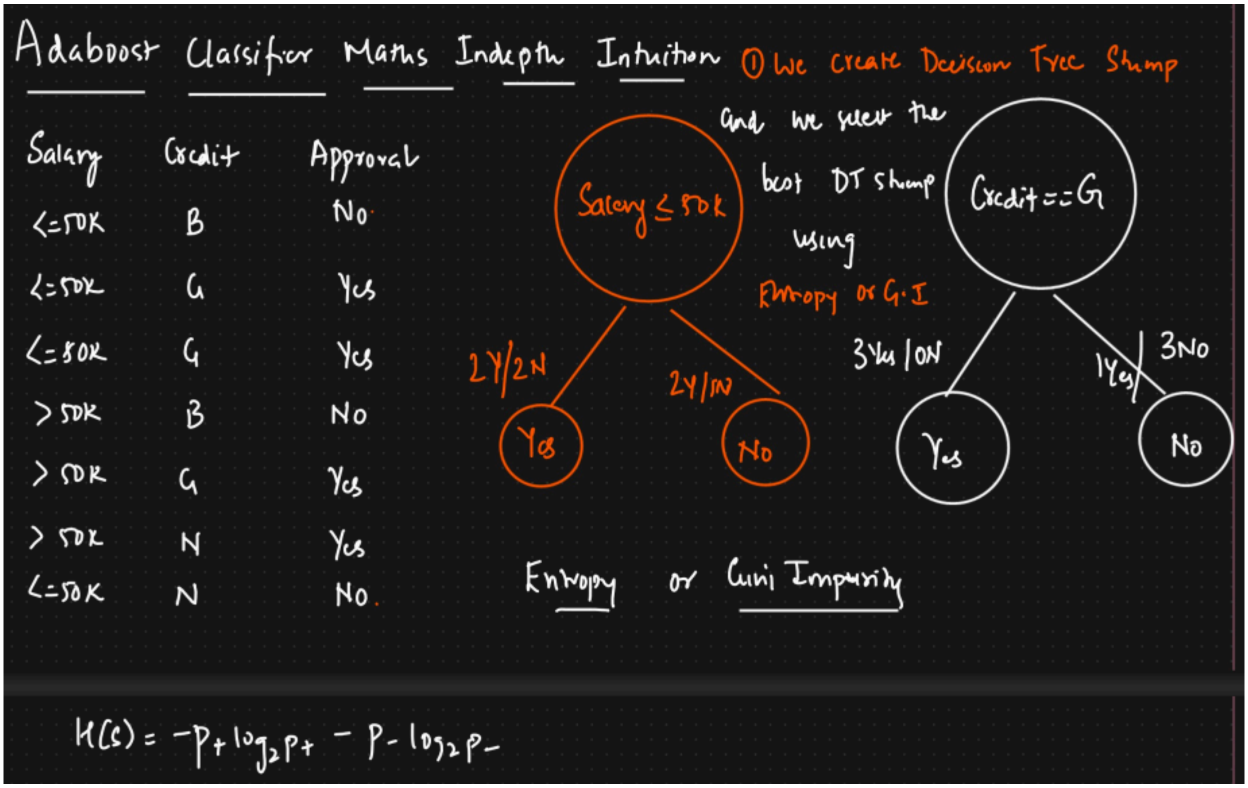
Random Forest → Majority Voting classifier
Average of (O/P)

↓

AdaBoost → Assignment weights to the weak learner



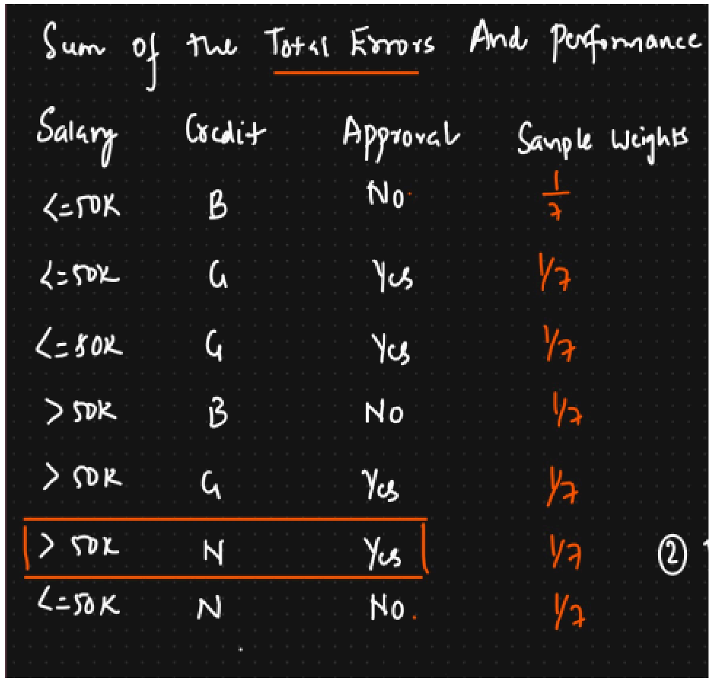
Decision Stump: Depth = 1.



3. How AdaBoost Works (Algorithm Steps)

Let's consider a binary classification problem (labels +1 and -1).

- **Initialization:** Assign equal weights to all data points in the training set. Let N be the number of data points, so each point starts with weight $w_i = 1/N.$.
- **Iterative Training (for M rounds/estimators):**
  - **a. Train a Weak Learner:** Train a simple model (e.g., a decision stump - a one-level decision tree) on the *weighted* training data. The goal is to minimize the weighted classification error.

- 

Sum of the Total Errors And Performance

Salary	Credit	Approval	Sample Weights	
<=50K	B	No	$\frac{1}{7}$	
<=50K	G	Yes	1/7	
<=50K	G	Yes	1/7	
>50K	B	No	1/7	
>50K	G	Yes	1/7	
>50K	N	Yes	1/7	2
<=50K	N	No	1/7	

- **b. Calculate Weighted Error (err_m):** Calculate the sum of weights of the misclassified points by the current weak learner (m).

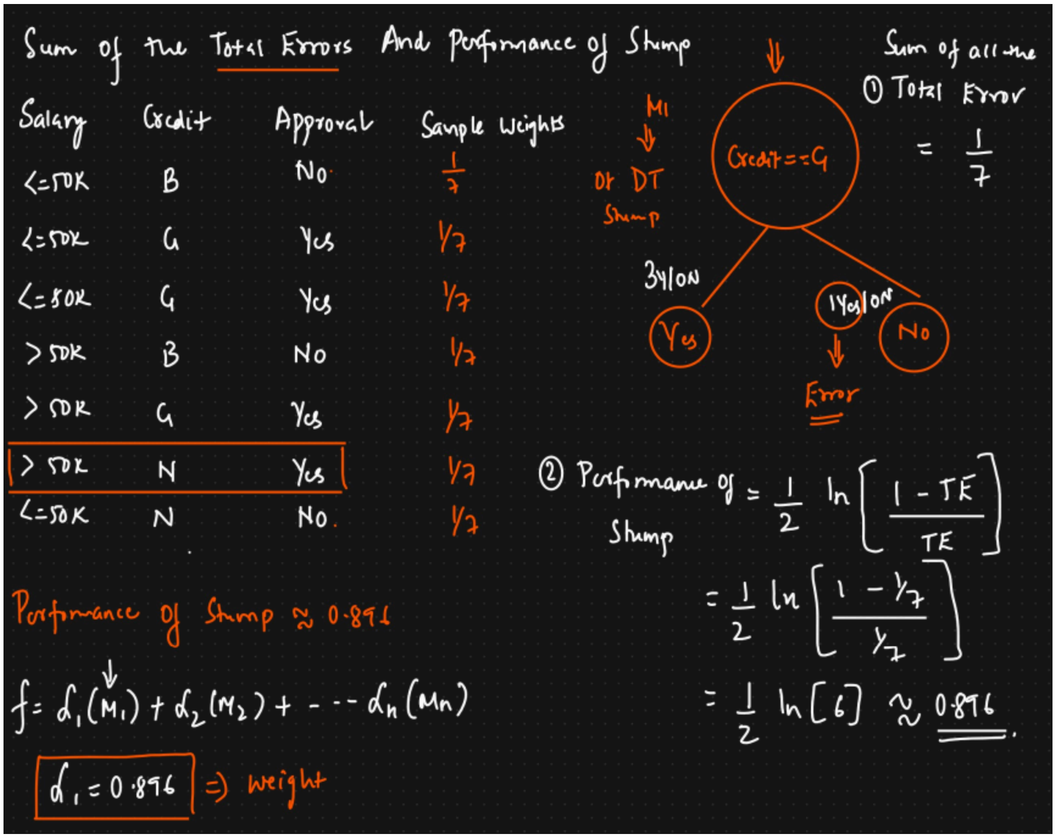
$$err_m = \frac{\sum_{i=1}^{N} w_{i,m} \cdot I(y_i \neq \hat{y}_m(x_i))}{\sum_{i=1}^{N} w_{i,m}}$$

  Where $w_{i,m}$ is the weight of sample i at iteration m, y_i is the true label, $\hat{y}_m(x_i)$ is the prediction of weak learner m for sample i, and $I(\cdot)$ is the indicator function (1 if true, 0 if false).

- **c. Calculate Learner Weight (α_m):** Assign a weight (importance) to this weak learner based on its error. Lower error means higher weight.

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - err_m}{err_m} \right)$$

-

- 

- **Error is 1/7 because 1 row is misclassified.**

d. Update Sample Weights ($w_{i,m+1}$): Increase the weights of misclassified samples and decrease the weights of correctly classified samples. This makes the "hard" samples more influential for the *next* weak learner.

$$w_{i,m+1} = \frac{w_{i,m} \exp(-\alpha_m y_i \hat{y}_m(x_i))}{Z_m}$$

Where y_i is the true label (+1 or -1), $\hat{y}_m(x_i)$ is the prediction (+1 or -1), and Z_m is a normalization factor (sum of all updated unnormalized weights) ensuring the new weights sum to 1.

- *Intuition:* If y_i and $\hat{y}_m(x_i)$ have the same sign (correct classification), the exponent is negative ($-\alpha_m$), reducing the weight. If they have opposite signs (misclassification), the exponent is positive (α_m), increasing the weight.

- (3) Update the weights for correctly and Incorrectly Classified points

Salary	Credit	Approval	Sample Weights	Update wts
<=50K	B	No	$\frac{1}{7}\downarrow \rightarrow$	0.058
<=50K	G	Yes	$1/7\downarrow \rightarrow$	0.058
<=50K	G	Yes	$1/7\downarrow \rightarrow$	0.058
>50K	B	No	$1/7\downarrow$	0.058
>50K	G	Yes	$1/7\downarrow$	0.058
>50K	N	Yes	$1/7\uparrow \rightarrow$	0.349
<=50K	N	No	$1/7\downarrow \rightarrow$	0.058

  For correct classified points

  $= \text{weight} * e^{-\text{Performance of stump}}$

  $= \frac{1}{7} * e^{-(0.896)}$

  $= 0.058$

  For Incorrect classified

  $= \text{weight} * e^{\text{Performance of stump}}$

  $= \frac{1}{7} * e^{(0.896)}$

- (4) Normalized Weights Computation And Assigning Bins

Salary	Credit	Approval	Update wts	Normalized weights	Bins Assignment
<=50K	B	No	0.058	0.08	0 - 0.08
<=50K	G	Yes	0.058	0.08	0.08 - 0.16
<=50K	G	Yes	0.058	0.08	0.16 - 0.24
>50K	B	No	0.058	0.08	0.24 - 0.32
>50K	G	Yes	0.058	0.08	0.32 - 0.40
>50K	N	Yes	0.349	0.50	$\boxed{0.40 - 0.70}$
<=50K	N	No	0.058	0.08	0.50 - 0.58
			0.697	≈1	

- **Iteratively select random values between 0 and 1 and chances of picking the point that is misclassified by the last learner is more.**

- 

  $\alpha_1 = 0.896$ Prepare datapoints

  (5) Select data points to send to Next Stump

Salary	Credit	Approval	Bins Assignment
<=50K	B	No	0 - 0.08
<=50K	G	Yes	0.08 - 0.16
<=50K	G	Yes	0.16 - 0.24
> 50K	B	No	0.24 - 0.32
> 50K	G	Yes	0.32 - 0.40
> 50K	N	Yes	$\boxed{0.40 - 0.50}$
<=50K	N	No	0.50 - 0.58

  (1) Iteration process selecting random value between 0 and 1

S	Credit	Approval	Random
>50K	N	Yes	0.50
<=50K	G	Yes	0.10
>50K	N	Yes	0.60
>50K	N	Yes	0.75
<=50K	G	Yes	0.24
>50K	B	No	0.32
>50K	N	Yes	0.87

- (6) These records will be sent to Next DT Stump.

S	Credit	Approval	Sample weight
>50K	N	Yes	1/6
<=50K	G	Yes	1/6
>50K	N	Yes	1/6
>50K	N	Yes	1/6
<=50K	G	Yes	1/6
>50K	B	No	1/6
>50K	N	Yes	1/6

  TE

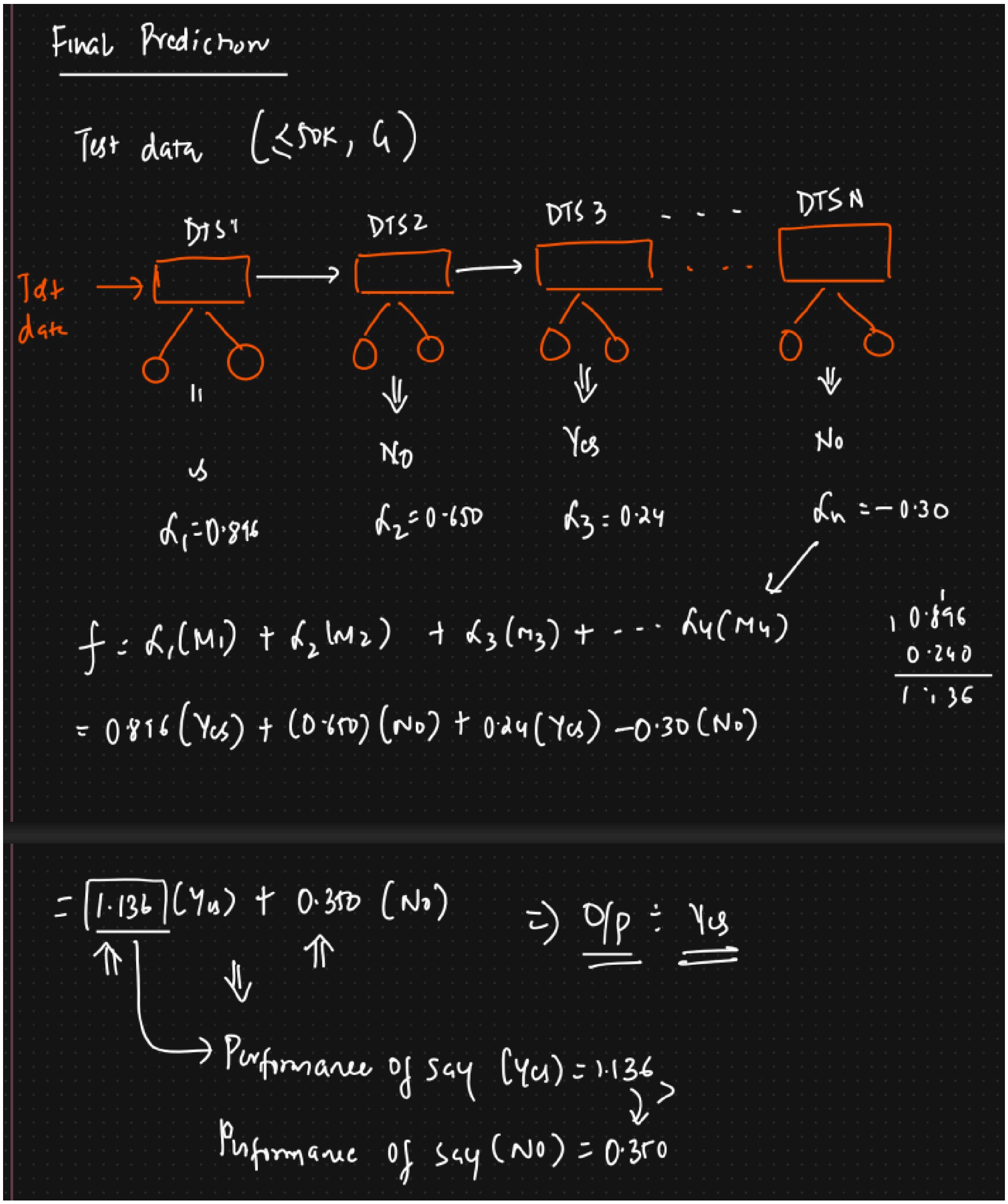
  Performance Stump ⇒ 0.65

  $$f_1 = \alpha_1(M_1) + \alpha_2(M_2)$$

  $\alpha_1 = 0.896$ $\alpha_2 = 0.65$

- **Final Prediction:** Combine the predictions of all M weak learners, weighted by their respective α_m values. The final prediction is the sign of the weighted sum:

$$\hat{Y}(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \hat{y}_m(x)\right)$$

- 

-

4. Key Components

- **Weak Learners:** Models that perform slightly better than random guessing (e.g., accuracy > 50% for binary classification). Decision stumps are very common, but other simple models can be used.
- **Sample Weights:** Indicate the importance of each data point during training. Adapts dynamically.
- **Learner Weights (α):** Indicate the contribution or "say" of each weak learner in the final prediction. More accurate learners get a bigger say.

5. Advantages of AdaBoost

- **High Accuracy:** Often achieves very good performance on classification tasks.
- **Less Prone to Overfitting (Theoretically):** While it can overfit, especially with noisy data, it's generally considered more resistant than some other algorithms, as the weak learners are simple. The number of estimators is a key parameter to tune.
- **Versatility:** Can be used with various types of weak learners.
- **Feature Importance:** Can provide insights into feature importance based on how often features are selected by the weak learners (especially when using decision stumps).

6. Disadvantages of AdaBoost

- **Sensitive to Noisy Data and Outliers:** Since AdaBoost focuses on misclassified points, outliers or noisy data can receive very high weights, potentially skewing the model. Preprocessing or using variants less sensitive to outliers might be needed.
- **Computationally Intensive:** Training is sequential, meaning models cannot be trained in parallel like in Bagging (e.g., Random Forests). Each step depends on the previous one.
- **Can be Complex to Tune:** Requires tuning parameters like the number of estimators and potentially the complexity of the weak learner.

7. Common Parameters (e.g., in Scikit-learn)

- `base_estimator`: The type of weak learner to use (default is usually DecisionTreeClassifier(max_depth=1) - a **decision stump**).
- `n_estimators`: The number of weak learners to train sequentially (M in the algorithm steps).
- `learning_rate`: Shrinks the contribution of each weak learner by this factor (values between 0 and 1). It helps prevent overfitting by making the model learn more slowly. This is technically a modification to the original AdaBoost algorithm but is standard in implementations. The weight update and final prediction formulas are adjusted slightly:
  - $\alpha_m = \text{learning\_rate} \times \frac{1}{2} \ln\left(\frac{1-err_m}{err_m}\right)$
  - $\hat{Y}(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \hat{y}_m(x)\right)$ (Formula stays the same, but α_m values are scaled)
-

8. Applications

- Primarily used for **binary classification** tasks (e.g., face detection - Viola-Jones algorithm, spam filtering, medical diagnosis).
- Can be extended to **multi-class classification** (variants like AdaBoost.M1, SAMME, SAMME.R).
- Can be adapted for **regression** tasks (variants like AdaBoost.R, AdaBoost.RT).