Boosting Algorithms

Reduce variance

Bagging : High variance, Low Bias + Randomization + aggregation

Base model

Boosting : Low variance, High Bias + additively combine

Reduce Bias while keeping Var. low

* High Bias can be viewed as high training errors.

Boosting is a method used to reduce errors in predictive data analysis.

Boosting is an ensemble learning method that combines a set of weak learners into strong learner to minimize training errors.

In Boosting, a random sample of data is selected, fitted with a model and then trained sequentially - i.e., each model tries to compensate for weakness of its predecessor. With each iteration, the weak rules from each individuals classifier are combined to form one, strong prediction rule.

Komal Diwe

Boosting algorithms can differ in how they create and aggregate weak learners during sequential process.

Three popular types of Boosting methods include:

- Adaptive Boosting or AdaBoost
- Gradient Boosting
- Extreme Gradient Boosting or XGBoost

XGBoost is an implementation of gradient boosting that's designed for computational speed and scale.

Komal Diwe

How is training in boosting done?

The training method varies depending on the type of boosting process called the boosting algorithm.

However, an algorithm takes the following general steps to train the boosting model:

Step 01

The boosting algorithm assigns equal weight to each data sample. It feeds the data to the first machine model, called the base algorithm.

The base algorithm makes predictions for each each data sample.

Step 02

The boosting algorithm assesses model predictions and increases the weight of samples with a more significant error.

It is also assigns a weight based on model performance. A model that outputs excellent predictions will have a high amount of influence

Komal Dive

over the final decision.

step 03

The algorithm passes the weighted data to the next decision tree.

step 04

The algorithm repeats steps 2 and 3 until instances of training errors are below a certain threshold.

Komal Diwe

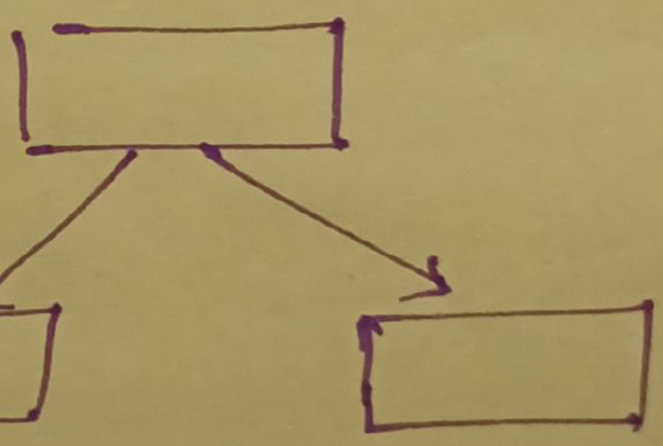
Adaptive Boosting

The three ideas behind adaBoost are

01. AdaBoost combines a lot of "weak learners" to make classifications.

The weak learners are almost aways stumps.

A tree with just one node and two leaves is called stump.



02. Some stumps get more say in the classification than others.

03. Each stump is made by taking the previous stump's mistakes into account.

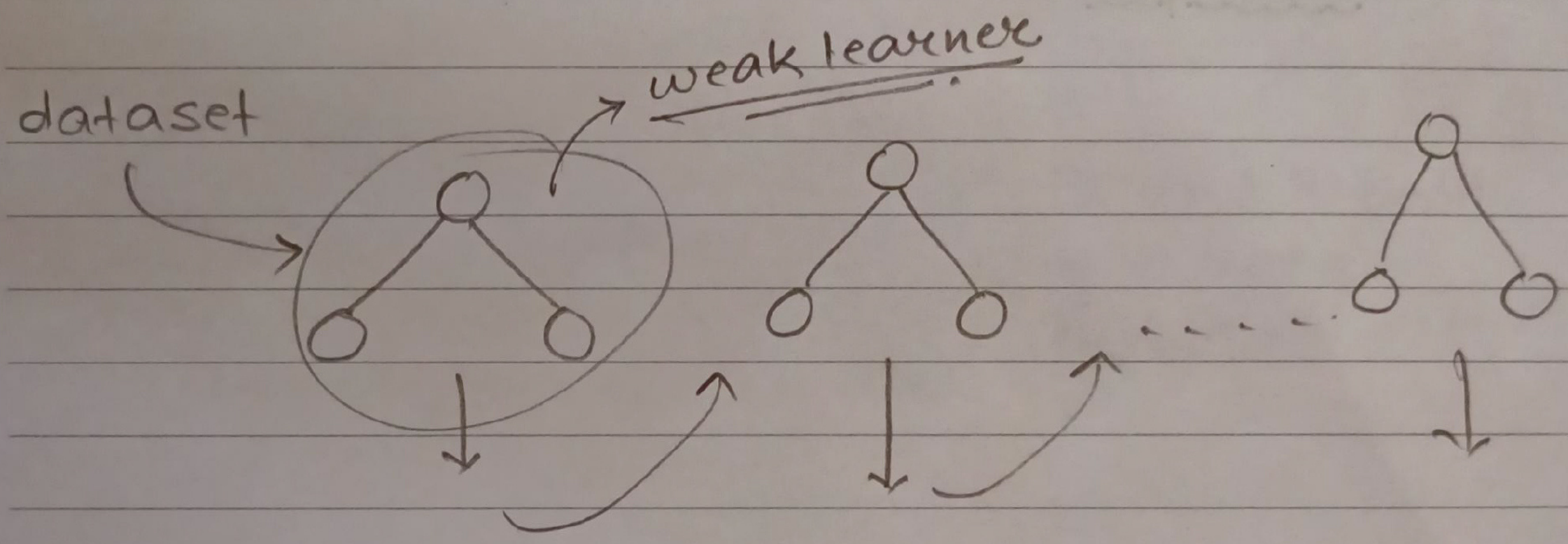
* Stumps are not great at making accurate classifications.

Komal Diwe

"

The most common algorithm used with AdaBoost is decision trees with one level that means with decision trees with only 1 split.

These trees are also called decision stumps.



Understanding the working of AdaBoost Algo.

In adaboosting, we combine many weak learners in sequence and every weak learner gives output by adding some weight and at the end we get strong learners.

Komal Diwe

$$f = \alpha_1(m_1) + \alpha_2(m_2) + \cdots + \alpha_n(m_n)$$

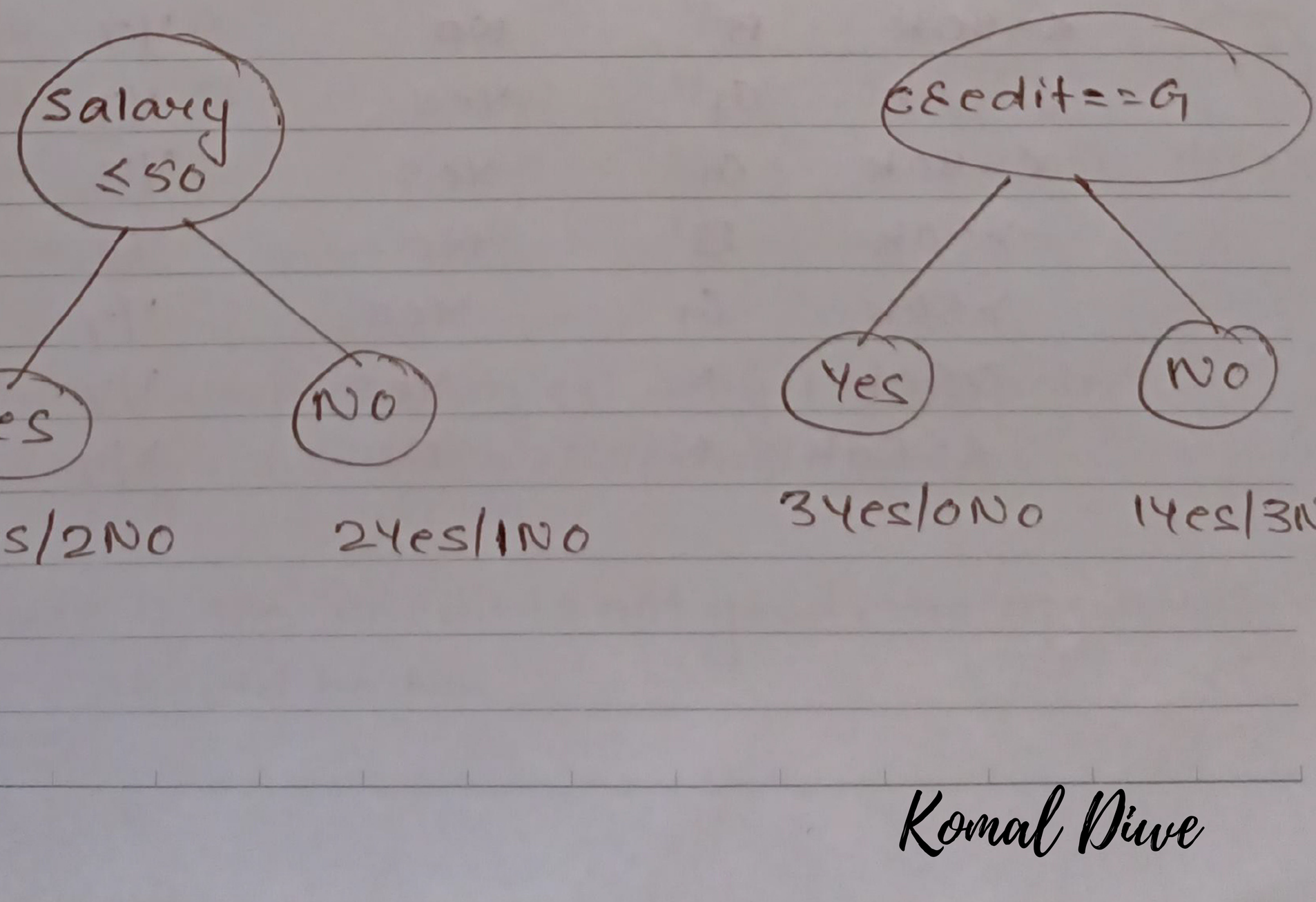
where, m = model $(m_1, m_2, \ldots, m_n)$

α = weight of every model

$(\alpha_1, \alpha_2, \ldots, \alpha_n)$

Example dataset:

Salary	Credit	Approval
<=50k	B	NO
<=50k	G	Yes
<=50k	G	Yes
>50k	B	NO
>50k	G	Yes
>50k	N	Yes
<=50k	N	NO



Komal Diwe

step 01: First of all these data points will be assigned some weights.

Initially, all the weights will be equal.

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \quad i = 1, 2, \ldots, n$$

where, N = total number of datapoints

Here since we have 7 datapoints so the sample weights assigned will be 1/7.

Salary	Credit	Approval	Sample weights
<=50k	B	NO	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
<=50k	N	NO	1/7

Komal Dive

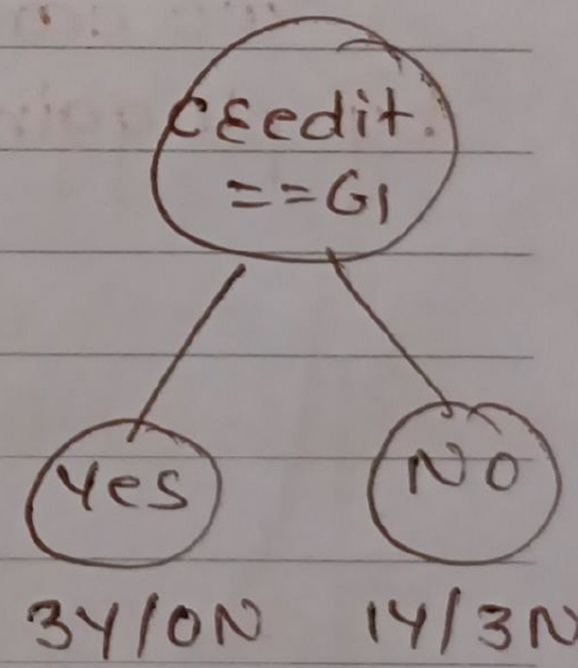
Step 02: We'll create a decision stump for each of the features and then calculate the Gini Index / Information Gain of each tree.

The tree with the lowest Gini Index / Highest Information Gain will be our first stump.

In our dataset, "Credit" has the lowest gini index so it will be our first stump.

B = Bad, G = Good, N = Normal

Salary	Credits	Approval	sample weight
<=50k	B	No	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
<=50k	N	No	1/7



This stump classified only 1 datapoint incorrectly, i.e. >50k, Normal Yes.

because If the credit is not good, the approval should be No.

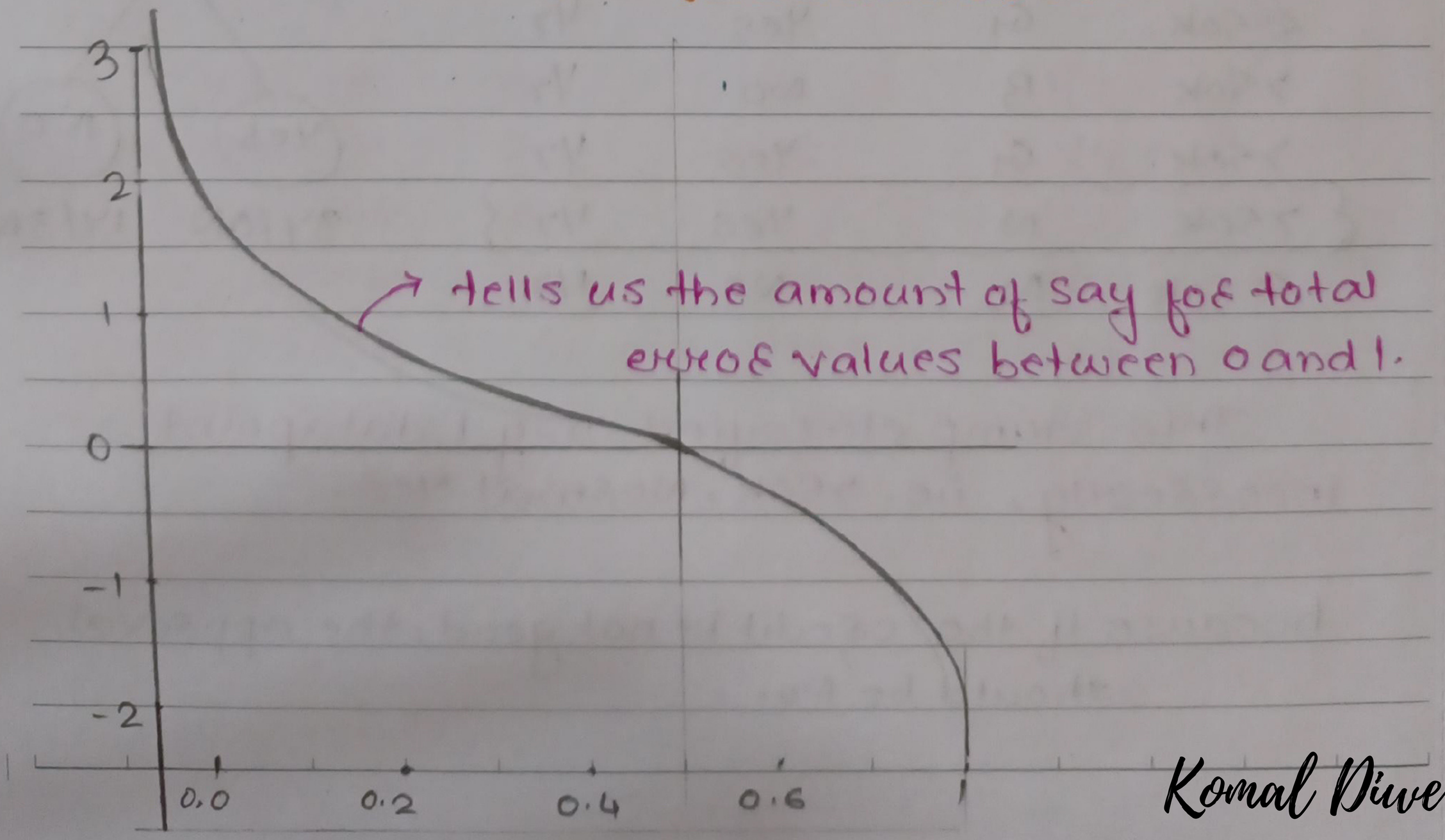
Komal Dive

step 03: we'll now calculate the "Amount of say" of "Importance" for this classifier in classifying the datapoints using formula:

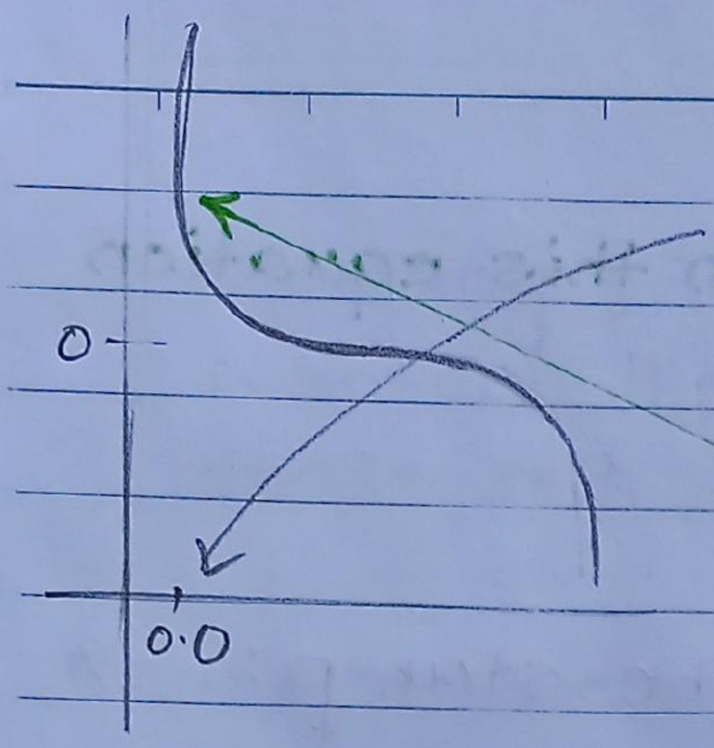
$$\text{Amount of say} = \frac{1}{2}\log\left(\frac{1-\text{total error}}{\text{total error}}\right)$$

The total error is nothing, but the summation of all the sample weights of misclassified datapoints.

we can draw a graph of the Amount of say by plugging in a bunch of numbers between 0 and 1 for total error.



Komal Diwe



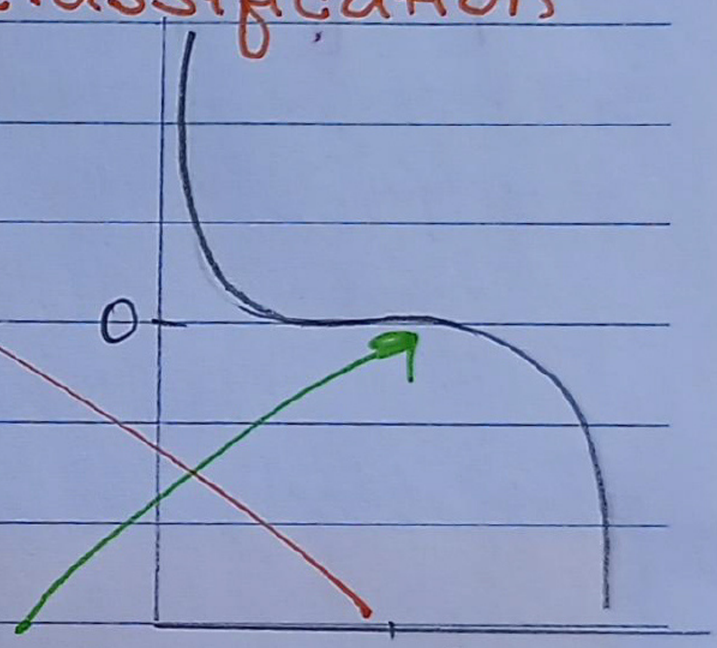
when a stump does a good job, and the total error is small

then the amount of say is a relatively large, positive value.

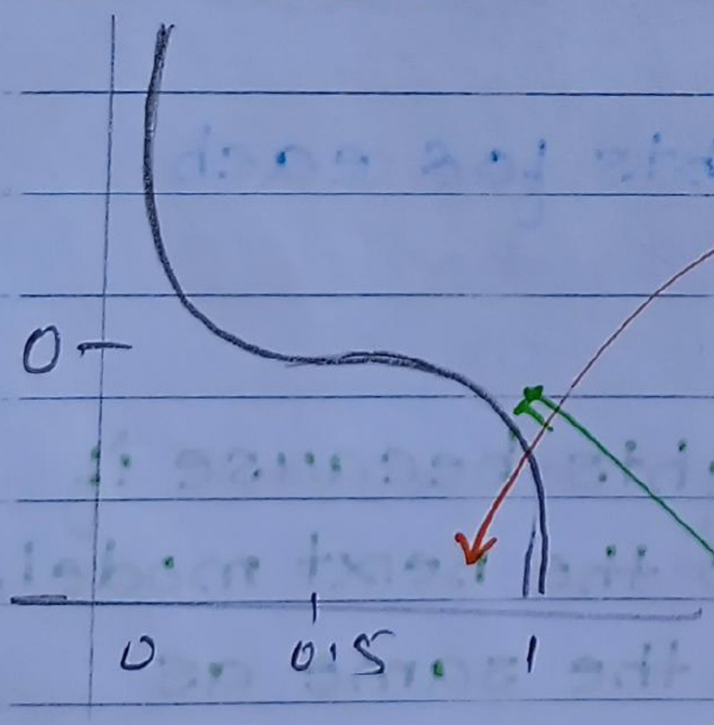
when a stump is no better at classification than flipping a coin

(i.e. half of the samples are correctly classified)

and Total Error = 0.5



then the amount of say will be 0.



and when a stump does a terrible job and total error is close to 1

in other words, if the stump consistently gives you the opposite classification

then the amount of say will be a large negative value.

Komal Diwe

Note: If Total Error is 1 or 0, then this equation will freak out:

Step 03:

Calculate the performance of the stump:

$$\text{performance of the stump} = \frac{1}{2}\log_e\left(\frac{1-\text{total error}}{\text{total error}}\right)$$

$$\alpha_1 = \frac{1}{2}\log_e\left(\frac{1-1/7}{1/7}\right)$$

$$= \frac{1}{2}.\log_e(6)$$

$$\alpha_1 = 0.895$$

Total Error will always be between 0 and 1.

Step 04: Calculate the new weights for each classified datapoint.

We need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model.

Komal Diwe

The wrong predictions will be given more weight whereas the correct predictions weights will be decreased.

Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.

After finding the importance of the classifier and the total error we need to finally update the weights and for this,
we use the following formula:

New sample weight = old weight * $e^{\pm \text{amount of say}(\alpha)}$

The amount of say (α) will be negative when the sample is correctly classified.

The amount of say (α) will be positive when the sample is miss-classified.

→ Increase the sample weight for incorrectly classified datapoints.

Komal Diwe

$$\therefore \text{new weight} = \frac{1}{7} \times e^{(0.895)}$$

$$= \underline{0.349}$$

decrease the sample weight for correctly classified datapoints

$$\text{new weight} = \text{old weight} \times e^{-\alpha}$$

$$= \frac{1}{7} \times e^{(-0.875)}$$

$$\therefore \quad \text{new weight} = 0.058$$

so the updated weight will be:

Salary	Credit	Approval	sample weight	updated weight
<=50k	B	NO	1/7	0.058
<=50k	G	Yes	1/7	0.058
<=50k	G	Yes	1/7	0.058
>50k	B	NO	1/7	0.058
>50k	G	Yes	1/7	0.058
>50k	N	Yes	1/7	0.349
<=50k	N	NO	1/7	0.058
			1	0.697

Komal Dive

__/__/__

Step 05: Normalize the sample weight.

If we add all the updated weights, we get 0.697 Hence, for normalization we divide all the sample weights by 0.697 and then create normalized sample weights as shown below:

Salary	Credit	Approval	updated weight	Normalized weight
<=50k	B	No	0.058	0.083
<=50k	G	Yes	0.058	0.083
<=50k	G	Yes	0.058	0.083
>50k	B	No	0.058	0.083
>50k	G	Yes	0.058	0.083
>50k	N	Yes	0.349	0.501
<=50k	N	No	0.058	0.083
			0.697	1

These new normalized weight will act as the sample weight for the next iteration.

step 06: Now repeat from step 02 and so on. till the configured number of estimates reached or the accuracy achieved.

Komal Diwe

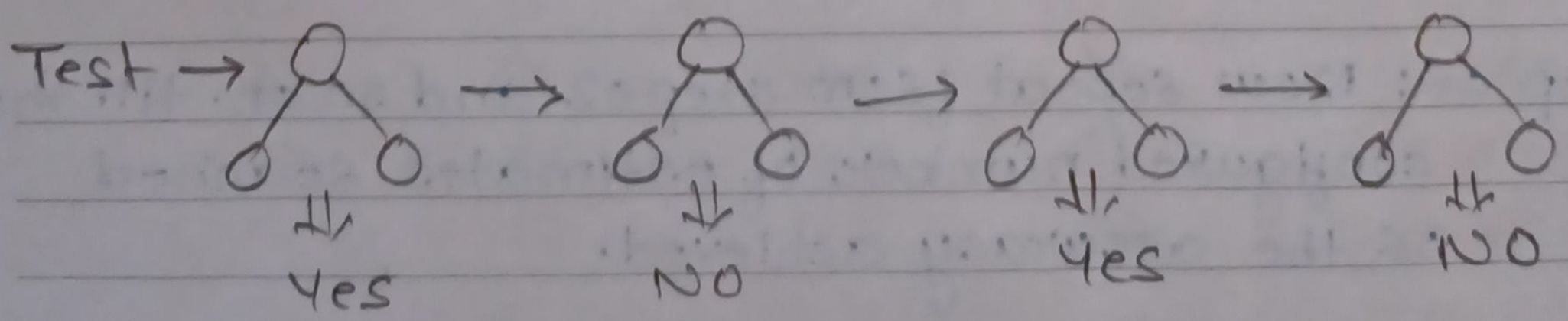
Now this will be the output of 1 decision tree; similarly we will receive output of many DT.

suppose, m trees (stumps) are classifying a person get approval "yes" and n trees (stumps) are classifying a person get approval "No", then the performance of the stumps (m trees and n trees) are added separately and whichever has the highest value, the person gets approval as that.

For example:

If the performance of m stump is 1.2 and the performance of n trees (stump) is 0.5 then the final result will go in the favour of m trees and the person will get the approval "Yes".

test (≤ 50k, G1)



Random assignments.

$\alpha_1 = 0.896$ $\alpha_2 = 0.650$ $\alpha_3 = 0.38$ $\alpha_4 = 0.20$

Komal Diwe

$$f = \alpha_1(m_1) + \alpha_2(m_2) + \alpha_3(m_3) + \alpha_4(m_4)$$
$$= 0.896(\text{Yes}) + 0.650(\text{No}) + 0.38(\text{Yes}) + 0.20(\text{No})$$
$$= 1.2(\text{Yes}) + 0.85(\text{No})$$

$\Downarrow$

Yes. (because weight of yes is more)

Komal Diwe