AdaBoost

AdaBoost short for Adaptive Boosting is an ensemble technique that combines multiple weak learners (models that are slightly better than random guessing i.e., their accuracy will be greater than 0.5) to create a strong learner with high predictive accuracy.
It is called adaptive because it focuses on correcting mistakes made by previous models in a sequential manner. Adaboost is a **stagewise additive method.**

* In context of classification problem, the weak learners should be slightly better than random guessing ie, their accuracy should be more than 0.5 whereas in context of regression problem, the weak learners should ideally produce predictions that are better than a simple average of the target variable.

→ **Stumps in AdaBoost :** A stump in AdaBoost is a very basic model with minimal decision-making capacity, designed to perform slightly better than random guessing. Due to its simplicity, a stump is often computationally inexpensive to train and thus allows the boosting algorithm to combine multiple such models / learners effectively. In general terms, a stump can be a weak learner / model that typically makes decisions based on a single feature.
We can have a stump of any algorithm like decision tree, linear regression, logistic regression, SVM and even neural networks given the weak learner satisfies the conditions mentioned above.

- **Decision Stumps :** Mostly we use decision trees as our weak learner, where the decision tree can split on only a single feature, thus the maximum depth this tree can have is 1 and it is known as decision stump.
- Each stump in Adaboost is made by taking previous stump's mistake into account. Some stumps get more say in prediction than others.

How AdaBoost works in Classification problems:

In Adaboost, for classification problem we have +1 and -1 class and not 0 class as in other ML algos.

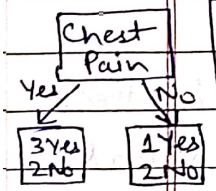
Let's start with a random dataset—

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Wi
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Step 1: Initialization → We start with equal weights for all training samples. We assign each sample equal weight $W_i = 1/n$, n = no. of samples.

Step 2: Make the first stump / train first weak learner → Since, we are going to use decision stumps, we can choose the feature for making the stump which gives us lowest impurity gini or highest information gain. This we have already studied in decision trees.

(a) Gini Impurity for Chest pain



Chest Pain	Heart Disease
Yes	Yes
No	Yes
Yes	Yes
Yes	Yes
No	No
No	No
Yes	No
Yes	No

$$G_{\substack{\text{chestpain} \\ \text{Yes}}} = 1 - \left[\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right] = 1 - \left[\frac{13}{25}\right] = \frac{12}{25}$$

$$G_{Yes} = 0.48$$

$$G_{No} = 1 - \left[\left(\frac{1}{3}\right)^2 + \left(\frac{2}{3}\right)^2\right] = 1 - \left[\frac{5}{9}\right] = \frac{4}{9}$$

$$G_{No} = 0.44$$

$$\therefore \text{Weighted } G_{chestpain} = \frac{5}{8} * 0.48 + \frac{3}{8} * 0.44 = 0.3 + 0.165$$

$$W_{Gini\ chestpain} = 0.465$$

(b) We do similar calculation for Blocked arteries.

Blocked Arteries	Heart Disease
Yes	Yes
Yes	Yes
No	Yes
Yes	Yes
Yes	No
Yes	No
No	No
Yes	No

Blocked Arteries → Yes: 3 Yes, 1 No; No: 1 Yes, 1 No

$G_{Yes} = 1 - [(\frac{3}{6})^2 + (\frac{3}{6})^2] = 1 - [\frac{1}{2}]$

$G_{Yes} = 0.5$

$G_{No} = 1 - [(\frac{1}{2})^2 + (\frac{1}{2})^2] = 1 - [\frac{1}{2}]$

$G_{No} = 0.5$

$WG_{Blocked\ Arteries} = \frac{6}{8} * 0.5 + \frac{2}{8} * 0.5$

$WG_{Blockedarteries} = 0.5$

(c) Similarly, we calculate Gini Impurity for Patient weight column — First, we sort the column in ascending order and then we calculate average of numerical values for consecutive numbers. Later, we can create different stages and calculate gini impurity for each stage, the stage whose gini impurity will be minimum will be chosen as the best stage for making a split amongst these.

	Patient Weight	Heart Disease	
140.5	125	No	
	156	No	161.5
	167	Yes	
167.5	168	No	
	172	No	170
176	180	Yes	
192.5	205	Yes	
	210	Yes	207.5

Weight > 176 → Yes: 3 Yes; No: 1 Yes, 4 No

$G_{Yes} = 1 - [(\frac{3}{3})^2] = 0$

$G_{No} = 1 - [(\frac{1}{5})^2 + (\frac{4}{5})^2] = 0.32$

$WG_{weight>176} = \frac{3}{8} \times 0 + \frac{5}{8} \times 0.32 = 0.2$

This gives us the least split among all numerical values.

Step 3: Now, we compare the gini impurity for each feature side by side and can see

$G_{weight\ 176} < G_{chestpain} < G_{Blockedarteries}$

Hence, Our decision stump will be made based on the Patient Weight feature.

Now, we need to determine how much say this stump will have in the final classification. We determine how much say a stump has in the final classification based on how well it classified the samples.

Weight < 176 → Yes HD: Correct 3, Wrong 0; No HD: Correct 4, Wrong 1

Thus our stump made one error for record in the table having row

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Wi
Yes	Yes	167	Yes	1/8

Here, the patient who weighs less than 176, has heart disease, but the stump says they do not.

The Total Error for a stump is the sum of weights associated with the incorrectly classified samples.
Thus, in this case the Total Error = 1/8.
Now, we can use this total error to determine amount of say this stump has in the final prediction with the following formula:

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

$$\Rightarrow \text{Amount of say } (\alpha_1) = \frac{1}{2}\log\left(\frac{1-1/8}{1/8}\right) = \frac{1}{2}\log 7 = 0.97$$

Step4: Now, we need to learn how to modify the weights so that the next stump will take the errors that the current stump made into account.
We will emphasize the need for the next stump to correctly classify the incorrect sample by increasing its sample weight (which was 1/8 in initial stump) and decreasing sample weights of others which were correctly classified.

New sample weight of incorrectly classified sample = sample weight $* e^{\text{amount of say}}$

$$= 1/8 * e^{0.97} = 1/8 * 2.64 = 0.33$$

This means new sample weight is 0.33, which is more than the old one i.e., 1/8 = 0.125
Now, we need to decrease the weights of correctly classified samples using the formula.

New sample weight of correctly classified samples = sample weight $* e^{-\text{amount of say}}$

$$= \frac{1}{8} * e^{-0.97} = \frac{1}{8} * 0.38 = 0.05$$

Here, new sample weight of correctly classified samples is 0.05 which less than the old one i.e., 0.125.

Step5:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	New Weight	Normalized weight
Yes	Yes	205	Yes	0.05	0.073
No	Yes	180	Yes	0.05	0.073
Yes	No	210	Yes	0.05	0.073
Yes	Yes	167	Yes	0.33	0.485
No	Yes	156	No	0.05	0.073
No	Yes	125	No	0.05	0.073
Yes	No	168	No	0.05	0.073
Yes	Yes	172	No	0.05	0.073
				sum=0.68	sum≈1

⇐ We keep track of new sample weights in the column, but since the weights needs to be normalized and should be

in range 0 to 1. We divide each sample weight by 0.68 which is the sum of all sample weights.

Step 6: Now, we need to upsample the samples from the dataset which were incorrectly classified, this is done as follows—

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Weights	Range	
Yes	Yes	205	Yes	0.073	0-0.073	
No	Yes	180	Yes	0.073	0.073-	0.146
Yes	No	210	Yes	0.073	0.146-	0.219
Yes	Yes	167	Yes	0.485	0.219-	0.704
No	Yes	156	No	0.073	0.704-	0.777
No	Yes	125	No	0.073	0.777-	0.85
Yes	No	168	No	0.073	0.85-	0.923
Yes	Yes	172	No	0.073	0.923-	1

Now, we pick 8 random numbers between 0 to 1 and the value which falls in the corresponding range we select that sample for our new dataset.

Let's say 8 such random numbers between 0 to 1 are:

0.512, 0.849, 0.034, 0.672, 0.189, 0.971, 0.428, 0.245

Corresponding rows → 4, 6, 1, 4, 3, 8, 4, 4

Our new dataset will be —

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	~~Weights~~	Weights
Yes	Yes	167	Yes	~~0.485~~	1/8
No	Yes	125	No	~~0.073~~	1/8
Yes	Yes	205	Yes	~~0.073~~	1/8
Yes	Yes	167	Yes	~~0.485~~	1/8
Yes	No	210	Yes	~~0.073~~	1/8
Yes	Yes	172	No	~~0.073~~	1/8
Yes	Yes	167	Yes	~~0.485~~	1/8
Yes	Yes	167	Yes	~~0.485~~	1/8

Now, we can observe the row which was misclassified by our stump 1, got upsampled 4 times showing its higher weightage.

Step 7: We then repeat the entire process again from begining on the new dataset and do continuously for all such decision stumps.

So, final prediction will be given as—

$$H(x) / p = \alpha_1 H_1(x) + \alpha_2 H_2(x) + \alpha_3 H_3(x) + \dots + \alpha_n H_n(x)$$

α_1: say of stump 1, α_2: say of stump 2, α_3: say of stump 3, α_n: say of stump n

* The final prediction is made by a weighted sum of the predictions of all the models. Models that perform better are given higher weight in the model.

→ In case of regression problems, we have error as squarred error i.e., $(y - y_{pred})^2$.

Bagging Vs. Boosting

	Bagging	Boosting
1)	In bagging, base models / weak learners used are having low bias and high variance. Eg. Fully grown decision trees.	1) In boosting, weak learners have high bias and low variance. Eg. Shallow decision stumps (having max-depth = 1).
2)	In bagging, all the models are trained in parallel and there is no interdependency.	2) In boosting, the models are trained in sequential manner, where current model depends on the previous model.
3)	In bagging, all the models have equal say in the prediction.	3) In boosting, all the models have different say in the final prediction.
4)	The primary goal of bagging is to reduce variance.	4) The primary goal of boosting is to reduce bias.
5)	In bagging, each model is trained on a randomly selected subset of the dataset, i.e., it is bootstrapped.	5) In boosting, initial stump / weak learner uses entire data but for the next stump the weights of the data points gets updated based on predictions made by initial stump. Misclassified samples are given more weight and correctly classified samples are given less weights, so the subsequent model focuses more on difficult cases.
6)	Eg. Random Forest	6) Ada Boost, Gradient Boost, XG Boost, Cat Boost, etc.