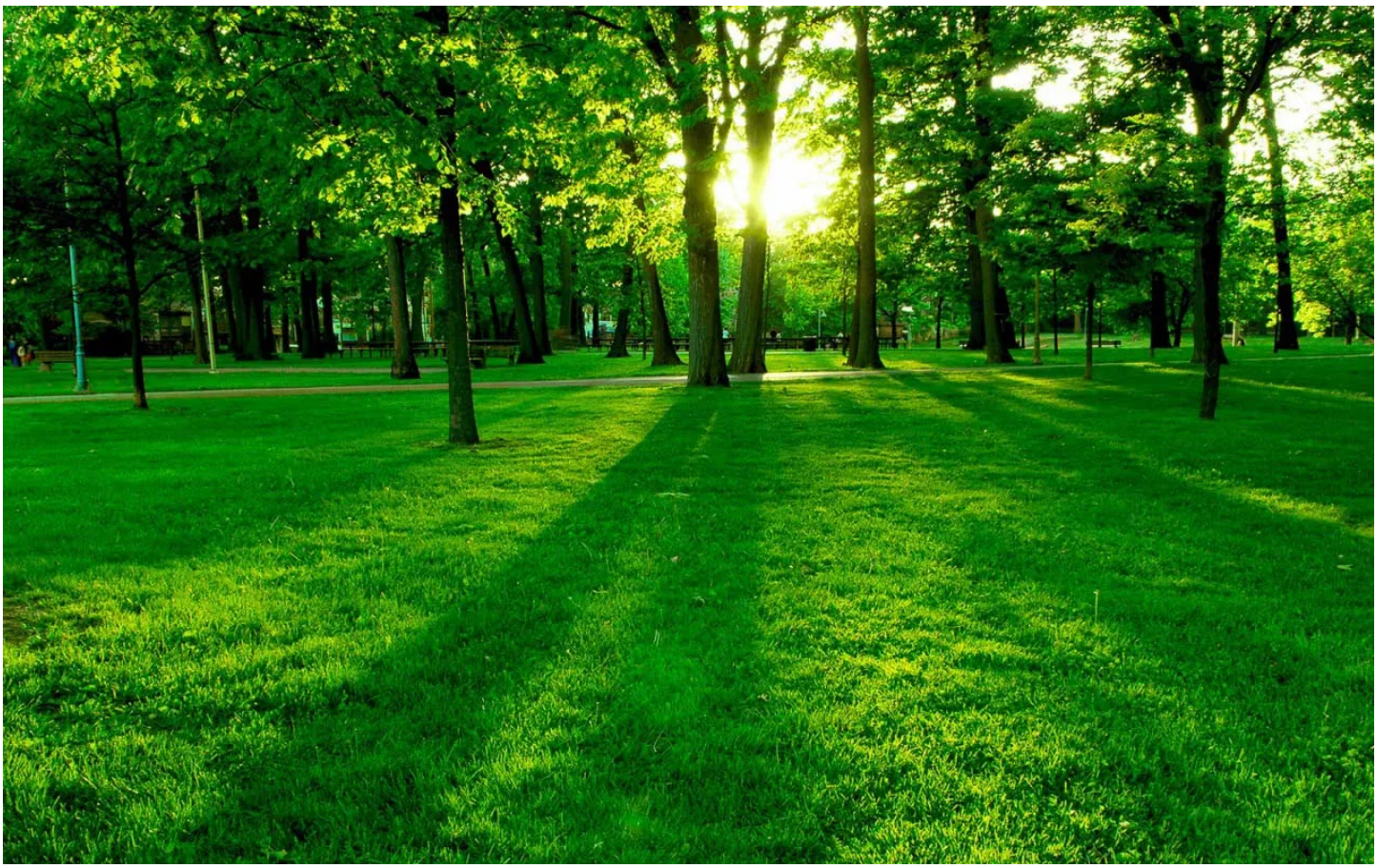
Adaptive Boosting or AdaBoost Algorithm.

karthikeyan thanigai · Follow

8 min read · May 26, 2020

In many Machine Learning models, the Algorithm will focus on high-quality prediction, But boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors. Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers.



In order to understand this Algorithm, you need to know about the basic concepts of decision trees and random forest, so go through this article:

Decision Trees and Random Forest — Just If-Else Repeatedly

When it comes to Machine Learning, we never forget to talk about Foresting algorithms as they have a very different...

medium.com

AdaBoost is best used to boost the performance of decision trees on binary classification problems. ***AdaBoost can be used to boost the performance of any machine learning algorithm.*** It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem. More recently it may be referred to as discrete AdaBoost because it is used for classification rather than regression.

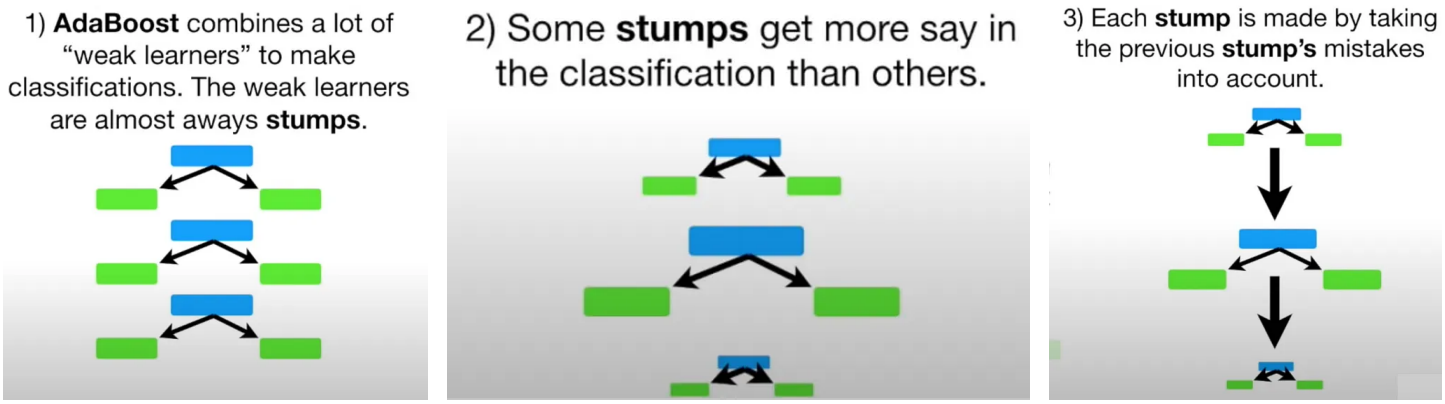
So in this article, We use the decision tree and random forest method's to understand the AdaBoost concepts. Then we will see how Adaboost creates a forest of trees and how it is used to make classification better than other methods.

So we will take this heart disease dataset for our example, so our main aim is to build a model that classifies whether a patient has heart disease are not when these independent variables (chest pain, blocked arteries, weight)are given.

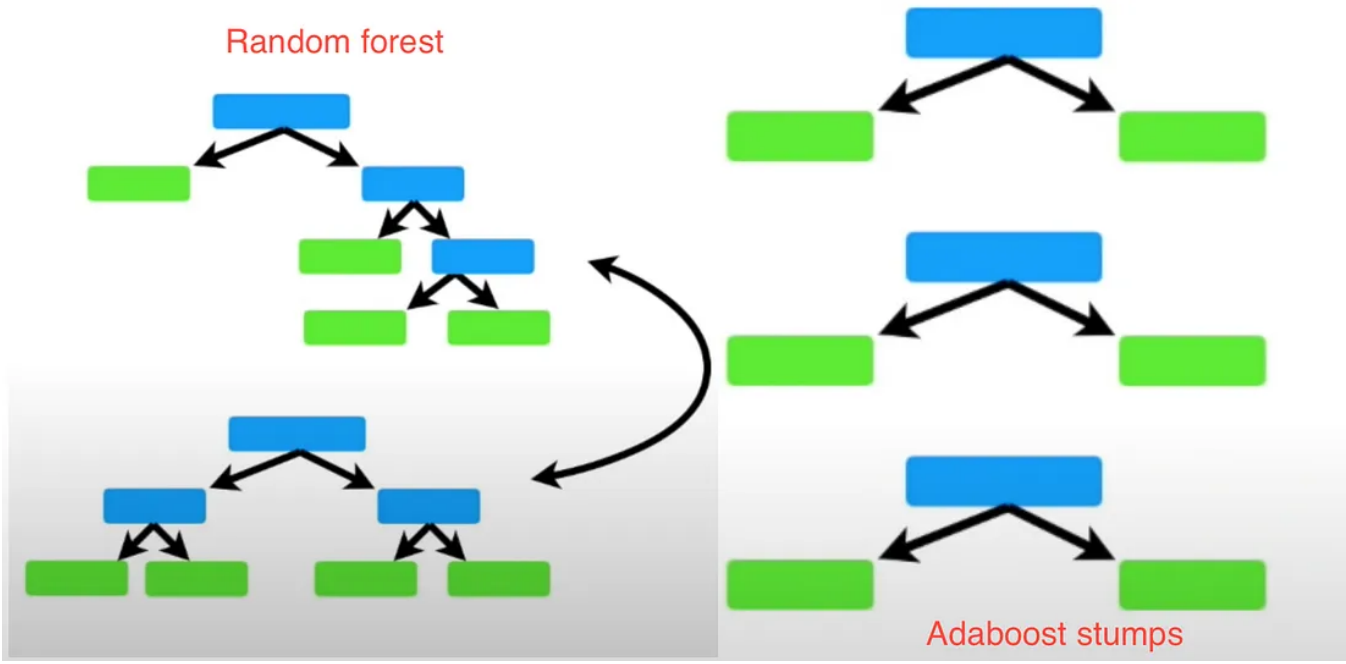
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

So now we will see the main 3 concepts of the Adaboost algorithm.

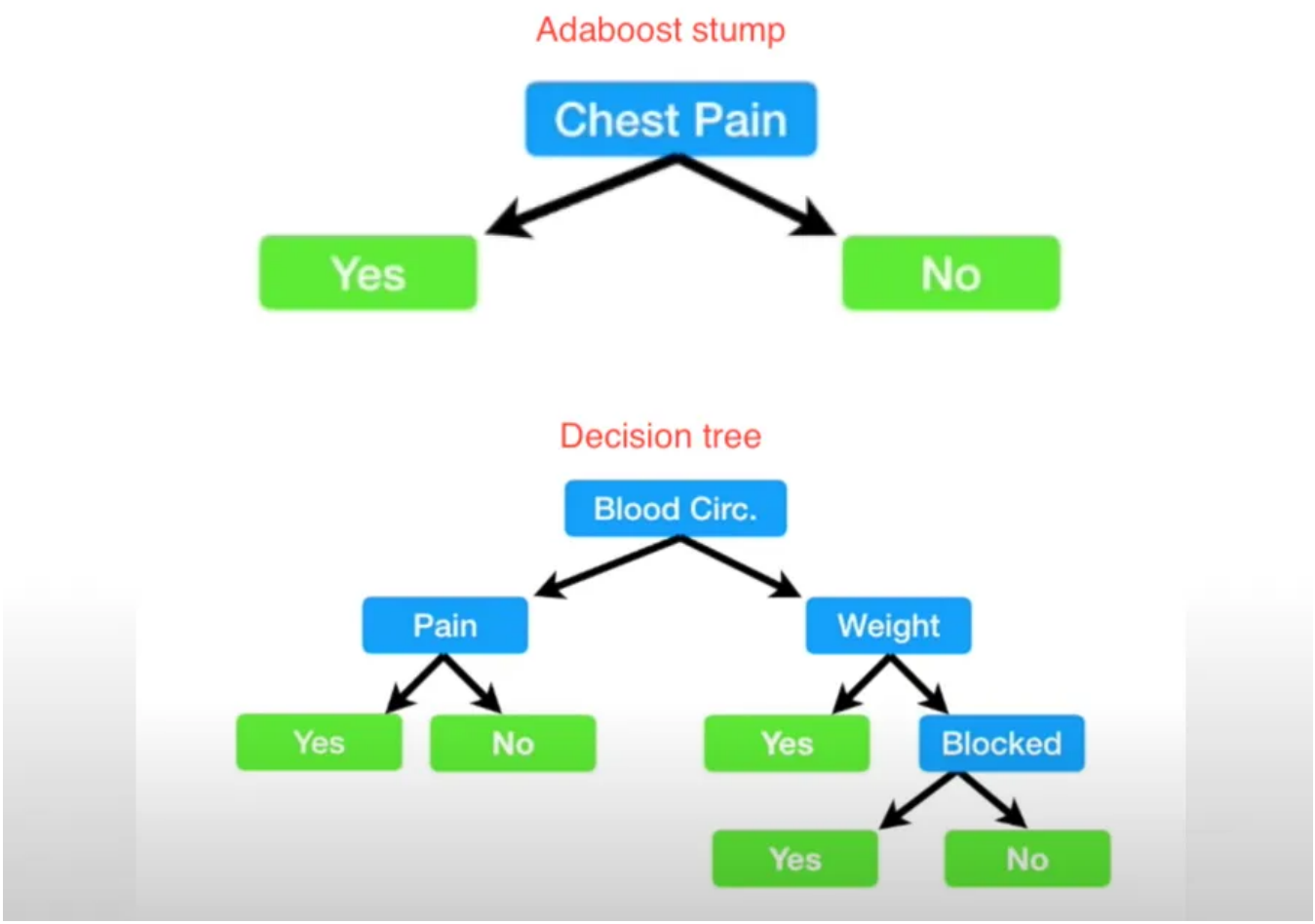
Adaboost main concepts:



Adaboost different concepts!!

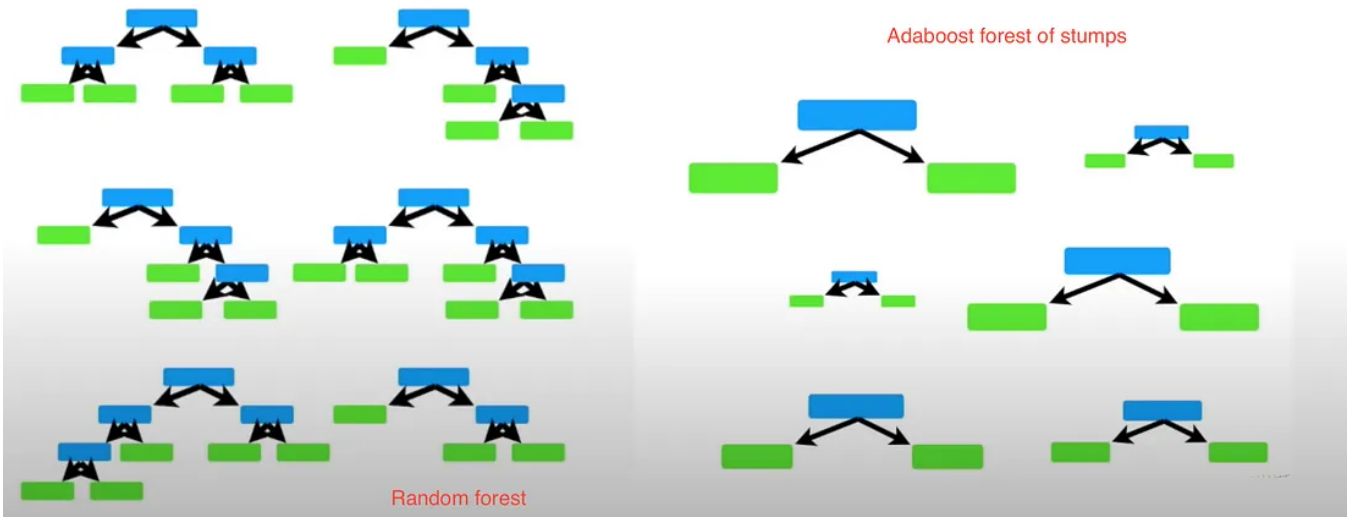


1). If you see in random forest method, the trees may be bigger from one tree to another but in contrast, the forest of trees made by ***Adaboost usually has just a node and two leaves.*** (A tree with one node and two leaves is called a stump)So Adaboost is a forest of stumps. stumps are not accurate in making a classification.



AdaBoost vs decision tree

for example, consider our heart disease dataset, a full-size decision tree we will take all the independent variables and form a tree so that the final predict depends on all the independent variable but stump can use only one variable to make a decision/predict, So there are weak learners but Adaboost use this stumps and finally ends up with a good result.



Random forest tree vs AdaBoost stumps

2). In a random forest, each tree has an equal vote on the final classification. But in Adaboost a forest of stumps, ***one has more say than the other in the final classification***(i.e some independent variable may predict the classification at a higher rate than the other variables)

3). Finally, in a random forest, each decision tree is independent of each other(i.e. one tree will not depend on other trees)*but in* ***Adaboost each stump depends on the predecessor's error*** (i.e.the error that the first stump makes will influence how the second stump...)

Adaboost implementation:

These are the main concepts of Adaboost now let's go deep in the process of implementation!!

Consider the above heart disease example, first what we will do is we will give some sample weights***(these values which will tell the importance of the sample value based on their error while predicting)*** that indicate how important it is to be correctly classified. At the beginning (for the first stump) all samples get the same weight value. that is,

Sample weight=1%No.of samples in the dataset

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
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

sample weight for our dataset

then we will find out the first stump, after the first stump, we will update the sample weight in order to make the next stump to predict that incorrectly classified sample correctly!!

Now we will find out the Gini index of all the independent variables in order to find out the first stump, lower the Gini index value is a good predictor of this class.

But what is this Gini index??

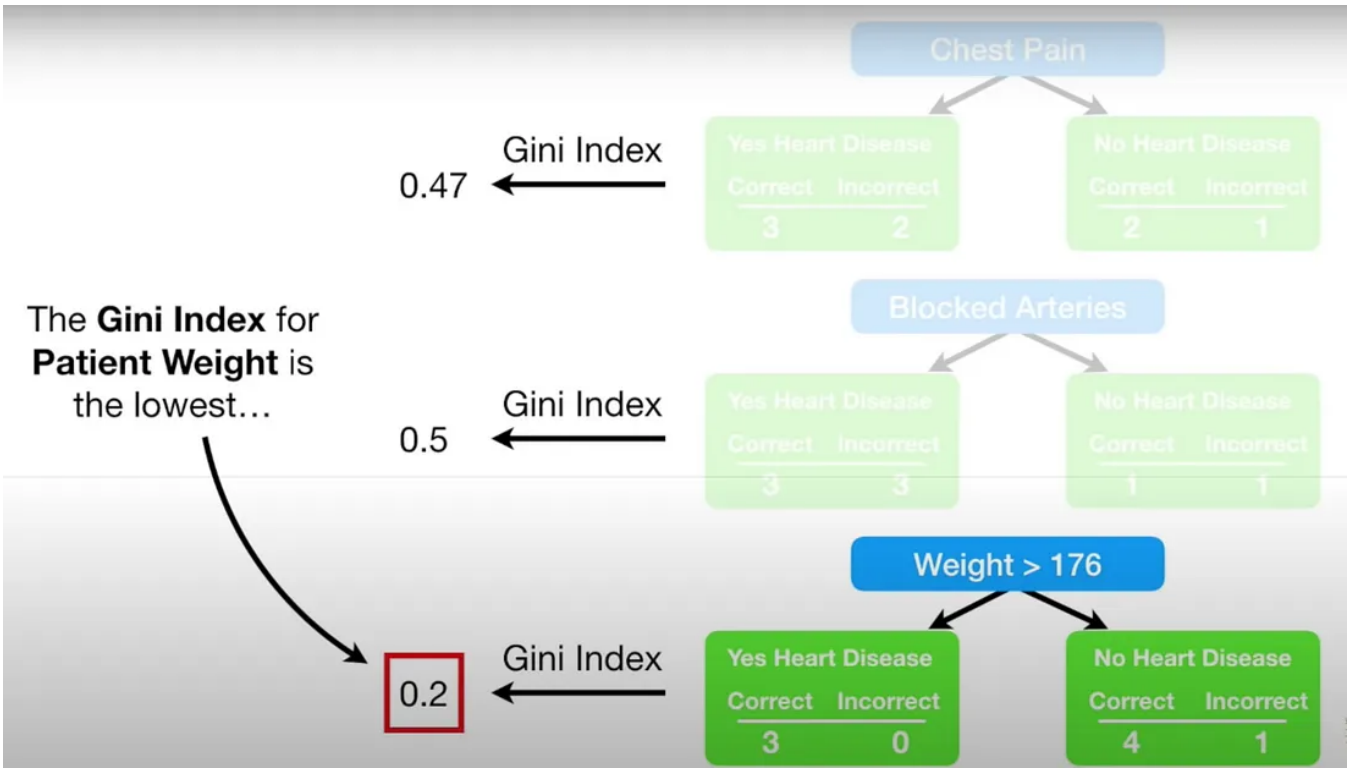
Gini index or Gini impurity measures the degree /probability of a particular variable being wrongly classified when it is randomly chosen. The degree of the Gini index varies between 0 and 1, where 0 denotes that all elements belong to a certain class or if there exists only one class, and 1 denotes that the elements are randomly distributed across various classes. A Gini index of 0.5 denotes equally distributed elements into some classes.

The formula for Gini Index

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

Gini index formula

where *pi* is the probability of an object being classified into a particular class.



So when we apply the Gini index to all the independent variable we get a low **Gini index of 0.2** to weight so we take the weight parameter as our first stump. the next least value is taken as the next stump...

Now we will find out how well this variable says to our class(how well this variable predicts) this is called **Amount of say.**

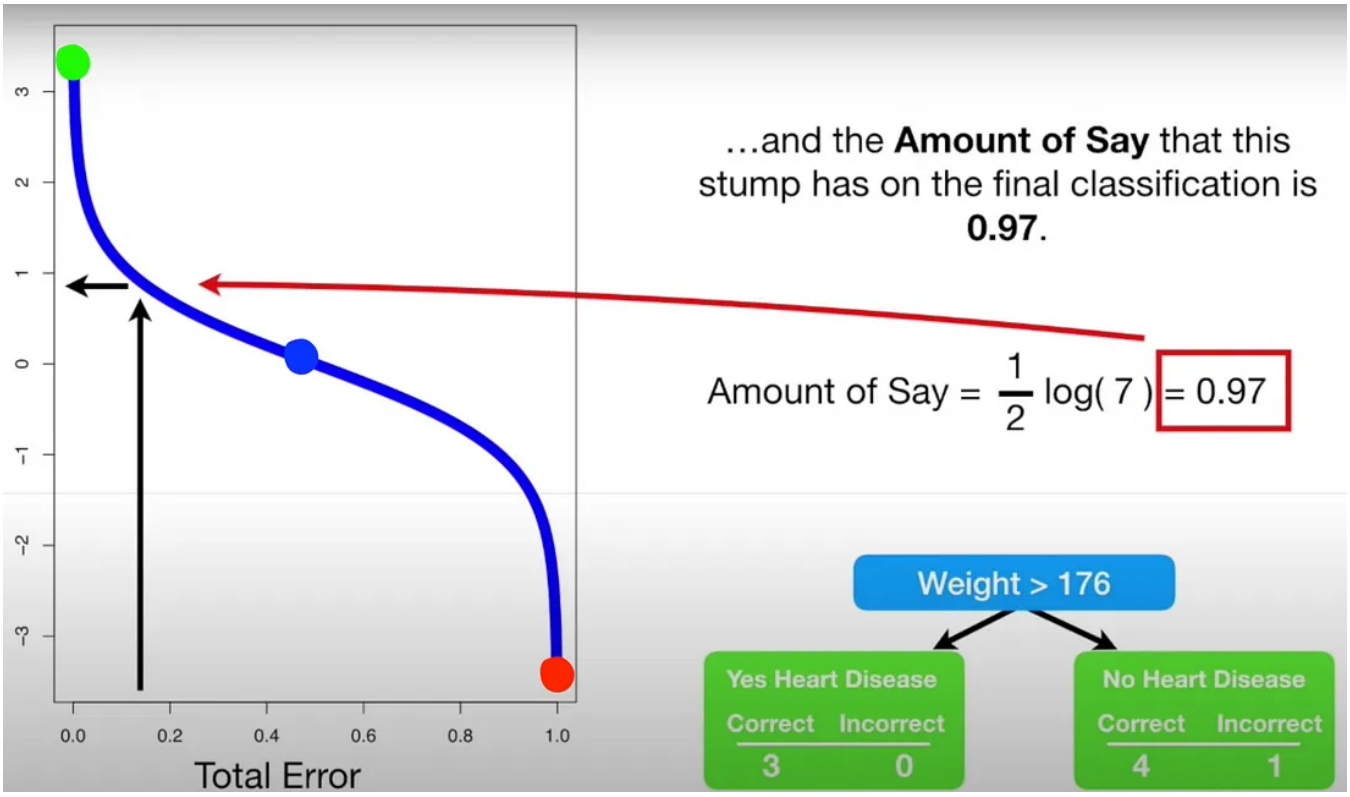
this stump made only one error, that is the patient whose weight is equal to 167(less than 176) has heart disease but our stump says that this patient does not have heart disease. we need to find out the **total error** did by our stump in order to say how well it predicts our data. ***So total error of a stump is the sum of weights associated with the incorrectly classified sample.***

So in our case the total error of weight stump is 1/8.***NOTE: because all of the sample weight add up 1, the total error will always be between 0, for a perfect stump, and 1 for a horrible stump!!***

We use the **Total Error** to determine **Amount of Say** this stump has in the final classification with the following formula:

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

Now we know the total error, So now we will find out how well this stump says to our class, to find out that use this formula in the image.



We can draw a graph that the amount of say by plugging in a bunch of numbers between 0 and 1 for total error. ***the blue tells us the amount of say for total error values between 0 and 1.*** when a stump does a *good job*, and the *total error is small*, then the amount of say is a relatively *large positive value(the green point)*. when the stump is *no better* at classification (i.e.half of the stumps are correctly classified and half are incorrectly classified) and *total error=0.5, then the amount of say will be zero(the blue point).* And when a stump does a *bad job* and the *total error is close to one* (in other words the stump predicts the wrong value all the time), then the amount of say will be a *large negative value(the redpoint)*. In our case the **amount of say=0.97** So it does a great job in predicting the final classification.

then after this first stump, we will update the sample weight (in order to predict that value correctly)and we will do this same process for all the other independent variables(that is we will take the next lowest Gini index value find out its total error and Amount of say and Accordingly change the sample weight). this process is repeated for all the independent variables and finally, the model will try to predict all the samples correctly, this is how Adaboost works!!

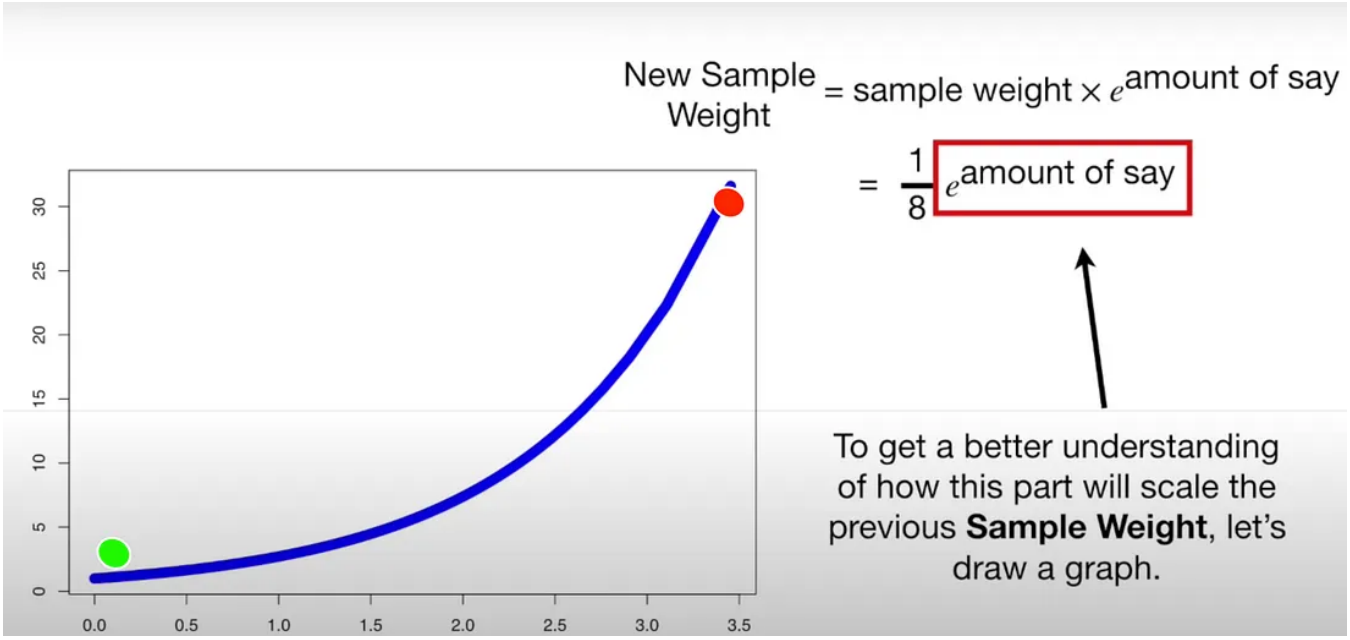
Now how to update the sample weight:

this is important because at the beginning all the Sample weight values are the same but after the first stump, we need to change this value in order to tell the next stump that *this sample was not correctly classified,* so the next stump will try to give more

preference to that sample(so at the end it can predict correctly that sample). So we will increase that sample weight value(which is been classified wrongly) and we will decrease all the other sample weights(which are classified correctly).

So the formula to find the new sample weight for the incorrectly classified sample is:

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$



To get a better understanding of how this part will scale the previous sample weight, let's draw a graph. ***the blue line is equal to e to the power amount of say,*** when the *amount of say is large*(i.e. the last stump did a good job in classifying samples) *then we will scale the previous sample weight value with a large value(*this means that the new sample weight will be much larger than the old one)*(redpoint)*. And *when the amount of say is relatively low*(i.e. the last stump did not do a good job in classifying samples)*then the previous sample weight is scaled by a relatively small value*(this means the new sample will be little larger than the old one)*(green point)*.In our case, the amount of say is 0.97 and the value of the **new sample weight is equal to 0.33**(which is more than the old one). this is how New sample weight value is found for the incorrectly classified sample.

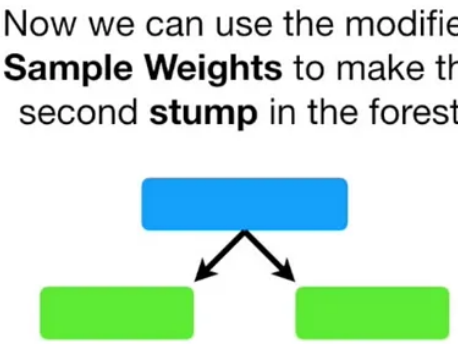
Now we will see the formula to find the new sample weight for the correctly classified sample is:

$$\text{New Sample Weight} = \text{sample weight} \times e^{-\text{amount of say}}$$

Same formula but just the negative sign. So the above formula is used to find out the new sample weight formula for the correctly classified sample. for our case, **the value is 0.05** which is less than the previous sample weight values.

So finally we update this weight value and Normalize our new sample weight so that they will add up to 1.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07



Updated Sample weights

So this is how Adaboost works!!

Here is the python implementation for Adaboost algorithm for classifying iris dataset:

So to finally Summarise:

In Adaboost we will first put a sample weight to our data and find out the Gini index for all the independent variables, then our first stump is found using the Gini index which is lower than all the other independent variables, then we will find the total error and amount of say for that stump.

then we will update the sample weight values for all the incorrectly classified samples and also for the correctly classified samples. then we will continue this for all the other stump...

this is how Adaboost works!!