Boosting

Philosophy of Boosting

- Additive models → **"Wisdom of Crowds"** → models that individually show poor performance can form a strong model when combined.
- Start with a very basic model (weak learner), we successively create more models and develop an ensemble system of multiple models (usually homogenous in nature)
- In boosting, a set of so-called weak learners, i.e., models whose performance is slightly better than random guessing, is built. The outputs of individual weak learners are combined as a weighted sum and represent the final output of the boosted classifier.
- Two major groups: AdaBoost and Gradient Boost

Adaboost

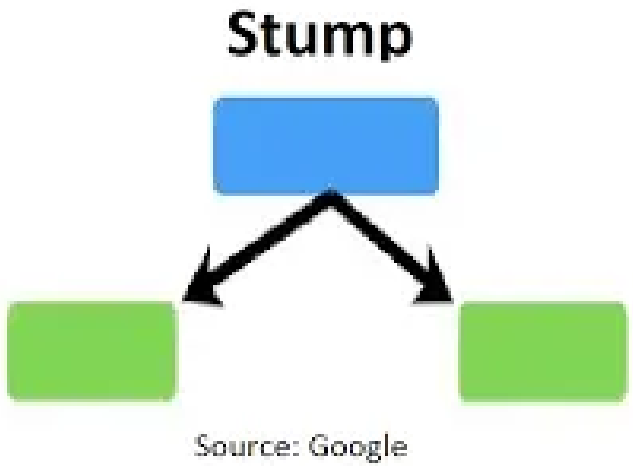
- AdaBoost is a Boosting algorithm based on Decision Tree
- AdaBoost stands for “Adaptive Boosting”. Adaptive because the models are built one after the other and the performance of the previous models influences the model-building process of the following models.
- During the learning process, the AdaBoost algorithm also assigns a weighting to each Weak Learner. Thus, not every weak learner has the same influence on the prediction of the ensemble model. This procedure for calculating the prediction of the overall model is called ***soft voting***. If, on the other hand, the results of each weak learner were weighted equally, we would speak of ***hard voting***.
- **Alpha1*HBLV1 + Alpha2*HBLV2 + Alpha3*HBLV3….. = LBLV model**

Bagging vs. Boosting

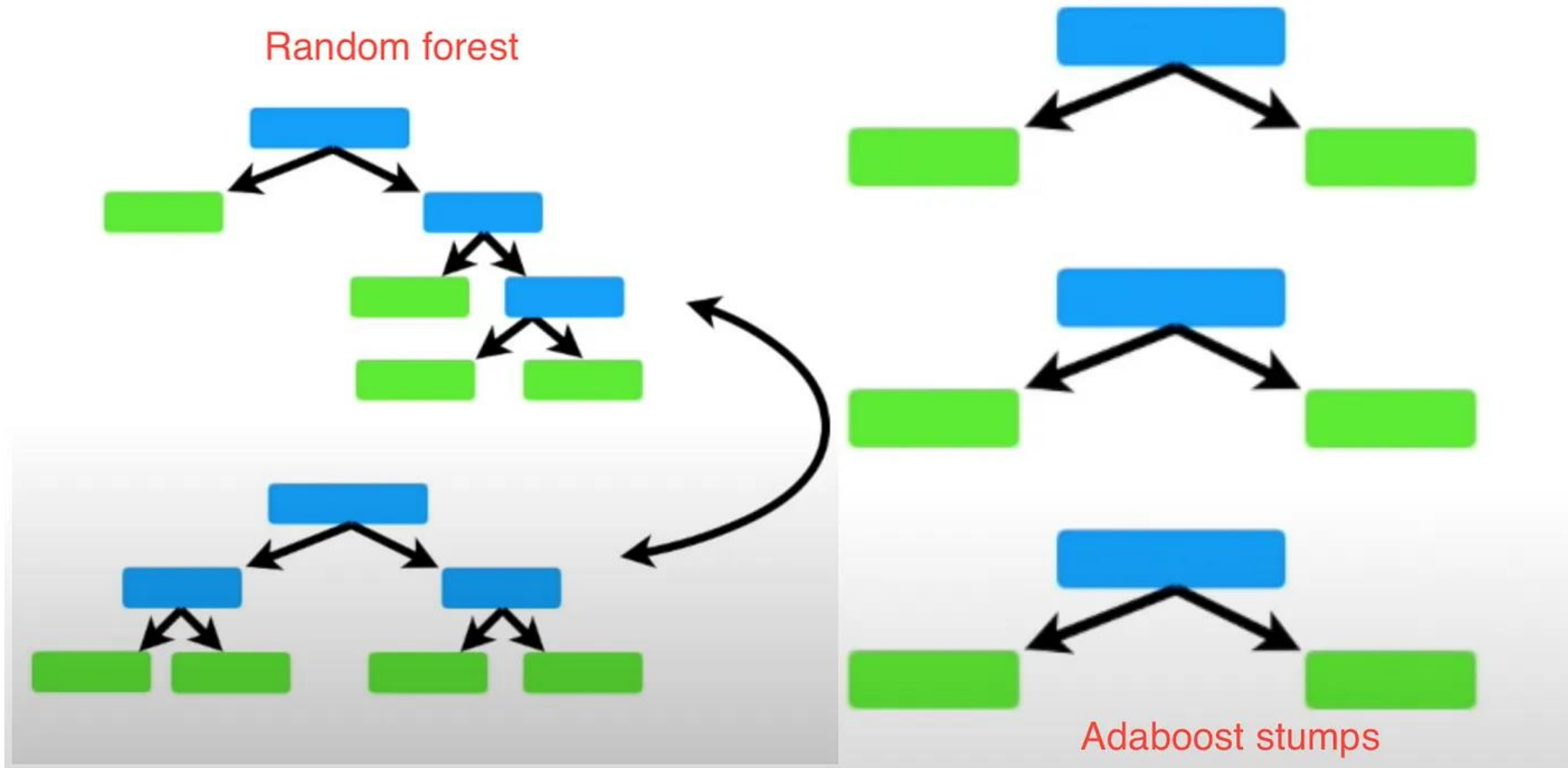
- In ***Bagging,*** we train a set of individual models that are independent of each other. The individual models differ from each other because they were trained with different, random subsets of the training dataset. The Random Forest is based on this principle. A set of individual decision trees form the prediction of the ensemble model.
- Training in ***Boosting***, on the other hand, is sequential. The model-building process of the individual models takes place one after the other, whereby the accuracy of the models predictions influences the training process of the subsequent models.

Random forest vs. Tree based AdaBoost

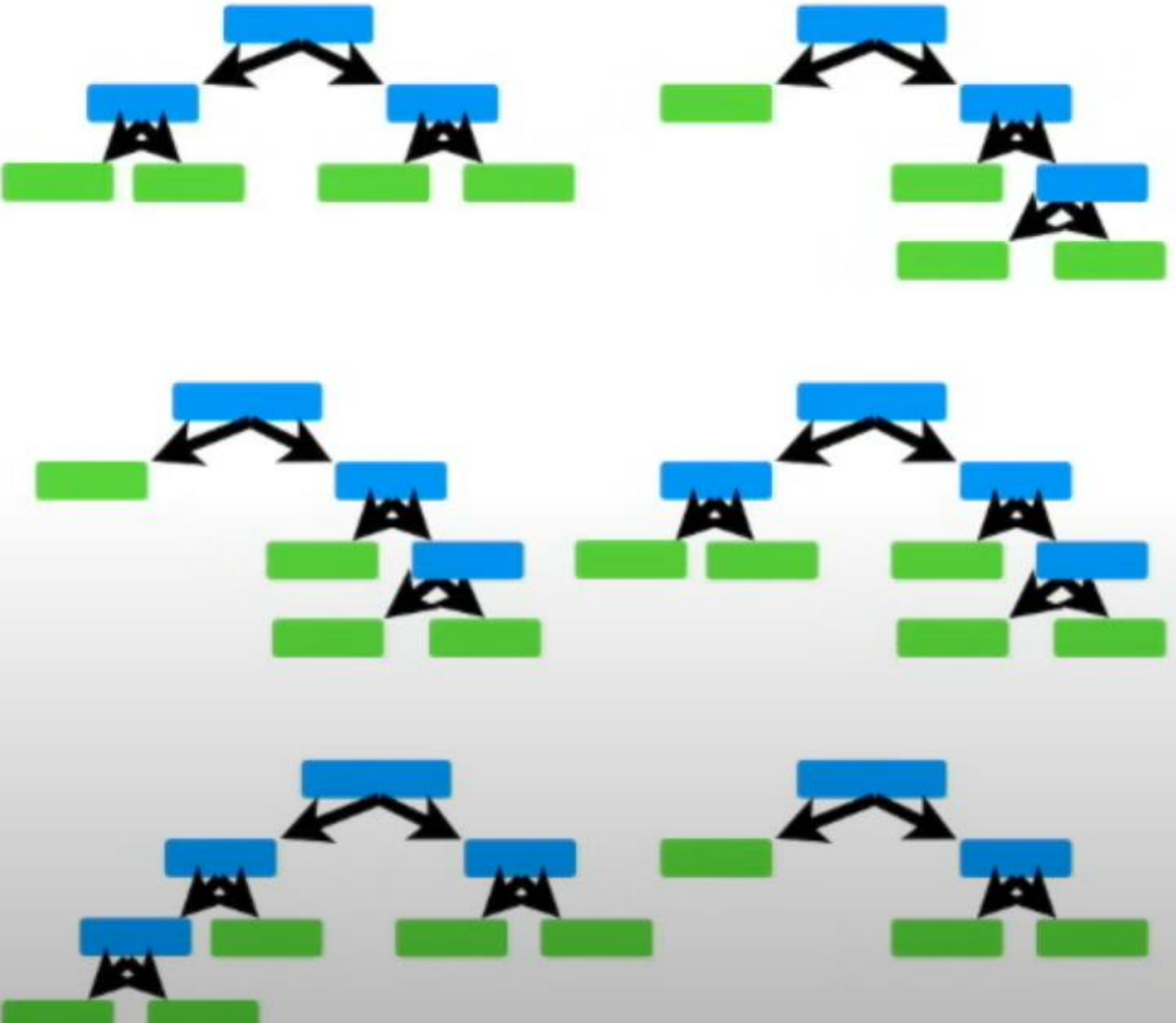
Random Forest is full of many different Decision Trees (fully grown). AdaBoost is built using a similar concept, but the algorithm uses stumps instead of trees.



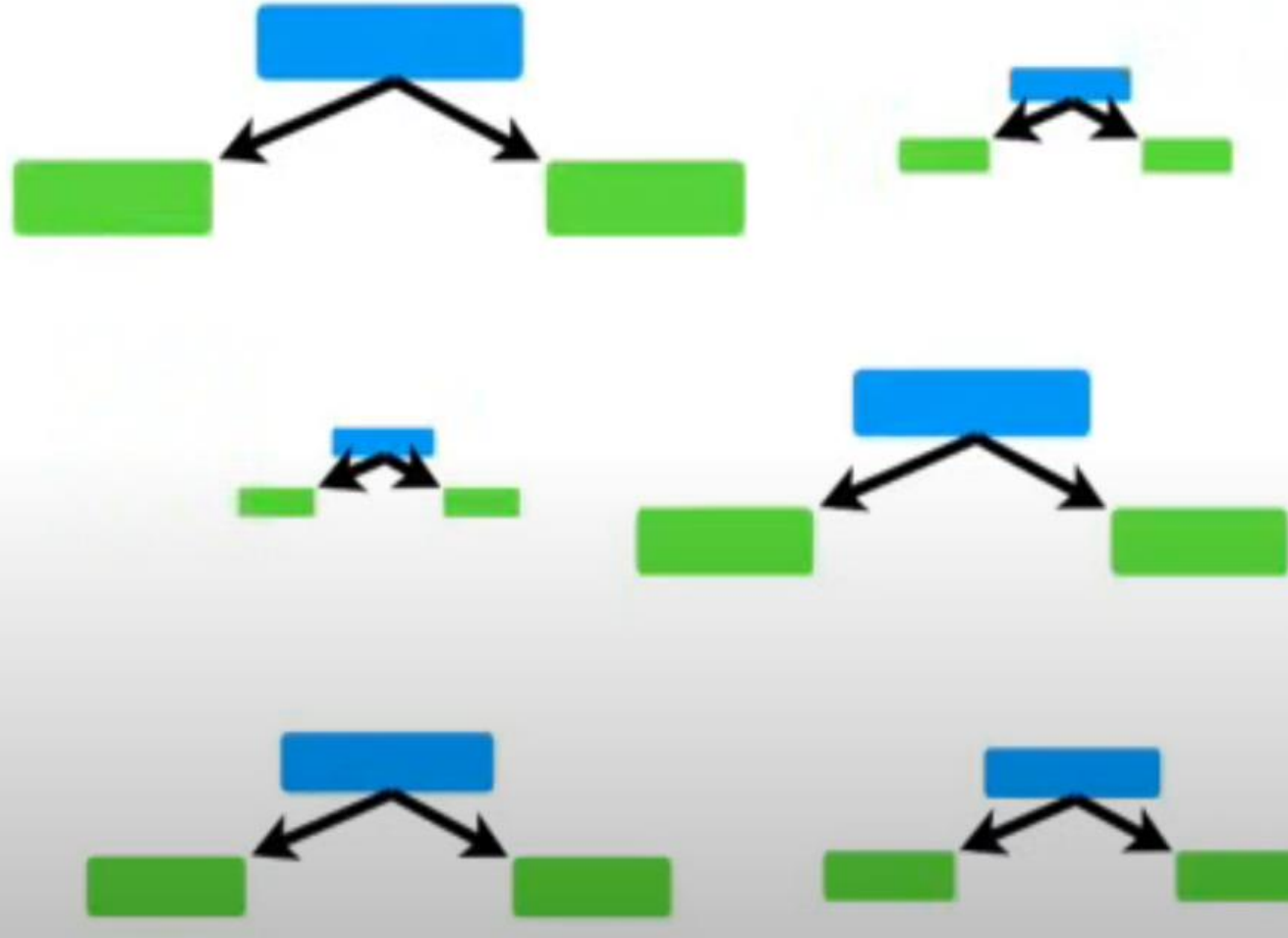
Basic component of Adaboost


Random forest
Adaboost stumps

In a **Random Forest**, each tree has an equal vote on the final classification.



In contrast, in a **Forest of Stumps** made with **AdaBoost**, some stumps get more say in the final classification than others.



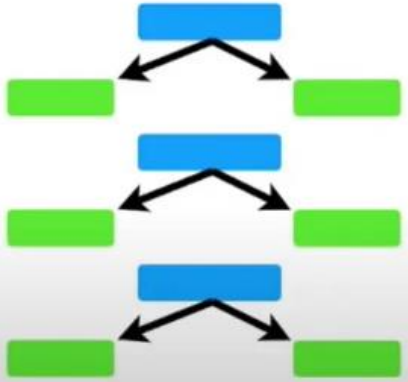
Lastly, in a **Random Forest**, each decision tree is made independently of the others.

The errors that the first stump makes...

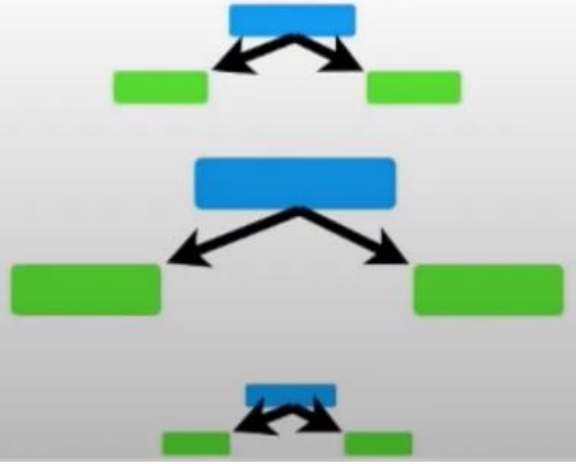
...influence how the second stump is made...

Adaboost main concepts

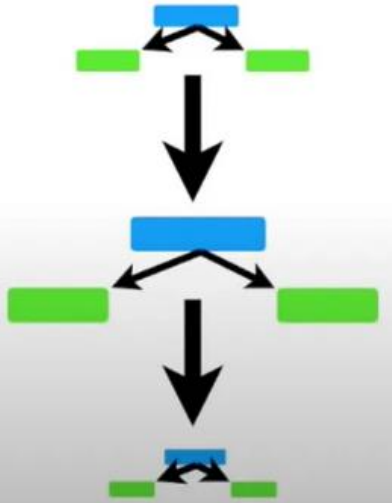
1) **AdaBoost** combines a lot of "weak learners" to make classifications. The weak learners are almost aways **stumps**.



2) Some **stumps** get more say in the classification than others.



3) Each **stump** is made by taking the previous **stump's** mistakes into account.



Additive models → aggregation of multiple weak learner

Algorithm AdaBoost

Input:

Training set $\{(x_1, y_1), ..., (x_n, y_n)\}$, where $x_i \in X$, $y_i \in \{-1, +1\}$
a weak learner algorithm
number of boosting rounds T
learning rate η

1: Initialize $D_1(i) = 1/n$ for $i = 1, ..., n$.
2: **for** $t = 1, ..., T$ **do**:
3: Train a weak learner using distribution D_t
4: Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$$

5: Choose

$$\alpha_t = \eta \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

6: Update, for $i = 1, ..., n$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor.
7: Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Adaboost algorithm steps

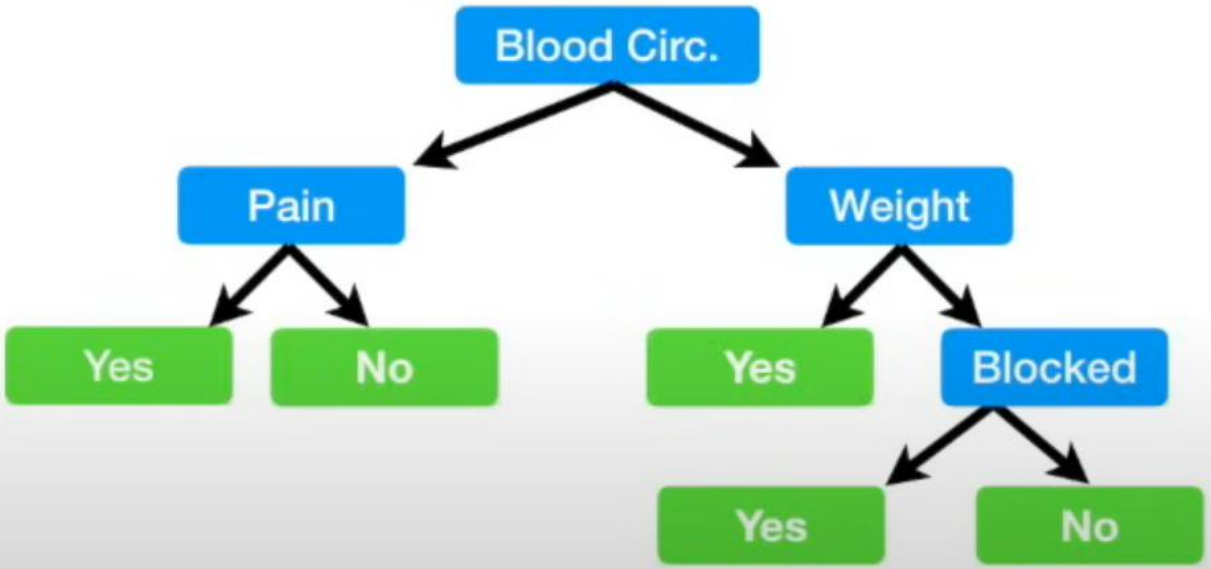
- **Step 1: Give the same weight to each sample of the dataset**
- **Step 2: Build the first Weak-Learner: Find the stump with the best performance i.e., provides maximum information gain (min. entropy/Gini index)**
- **Step 3: Calculate the misclassification error of the stump (on the training data)**
- **Step 4: Calculate the weighting of the stump, the "amount of say"**
- **Step 5: Modify the weights of the data samples in the dataset and normalize the weights**
- **Step 6: Create a new dataset by upsampling the data points (rows) which were misclassified by the created stump. Form a new Stump on the this new dataset**
- **Step 7: Repeat the process until the termination condition is reached**
- **Step 8: Determine the final prediction using decisions of all the decision stumps**

We will make these predictions based on a patient's **Chest Pain** and **Blocked Artery** status and their **Weight**.

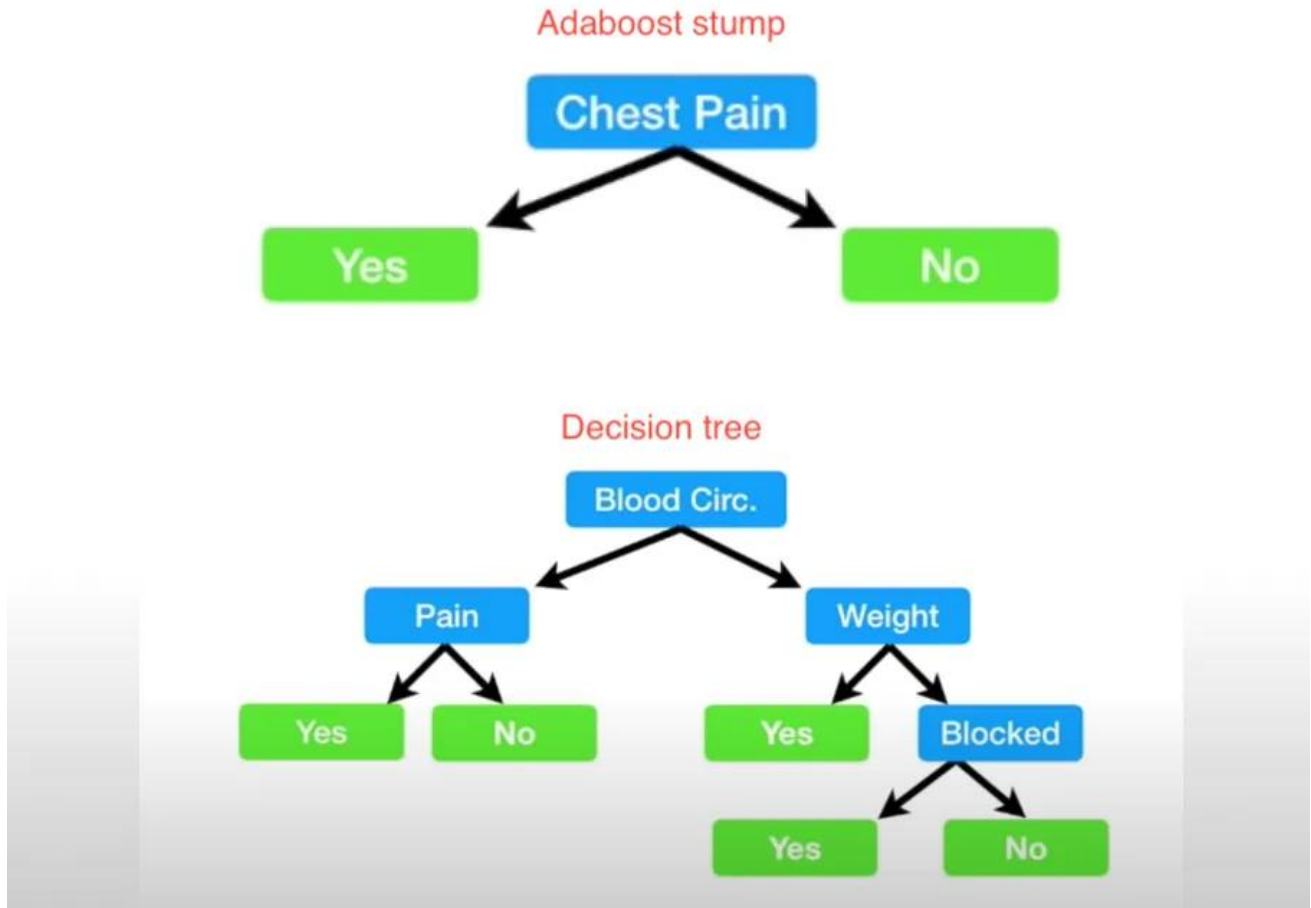
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

…then a full sized **Decision Tree** would take advantage of all **4** variables that we measured (**Chest Pain**, **Blood Circulation**, **Blocked Arteries** and **Weight**) to make a decision…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

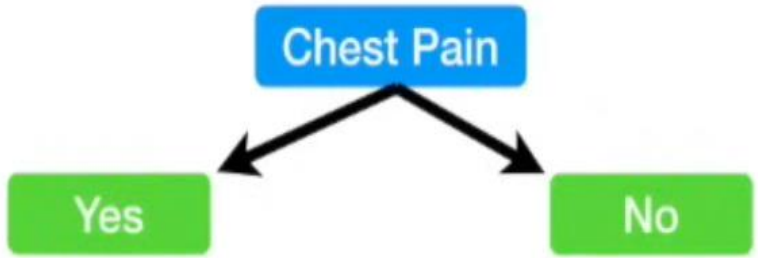


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

…but a **Stump** can only use one variable to make a decision.



Thus, **Stumps** are technically "weak learners".

However, that's the way **AdaBoost** likes it, and it's one of the reasons why they are so commonly combined.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
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	

The first thing we do is give each sample a weight that indicates how important it is to be correctly classified.

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

At the start, all samples get the same weight…

$$\frac{1}{\text{total number of samples}} = \frac{1}{8}$$

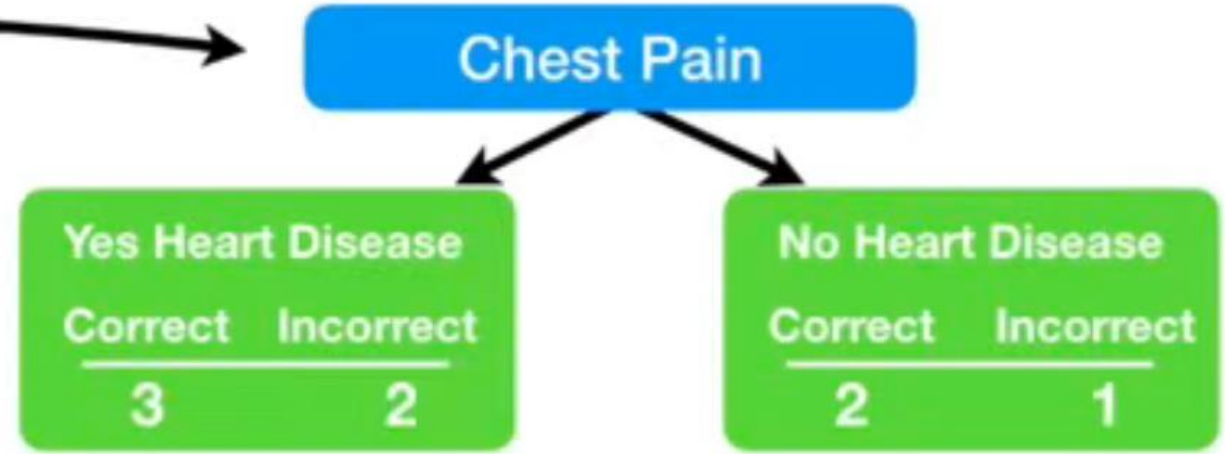
…and that makes the samples all equally important.

However, after we make the first stump, these weights will change in order to guide how the next stump is created.

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

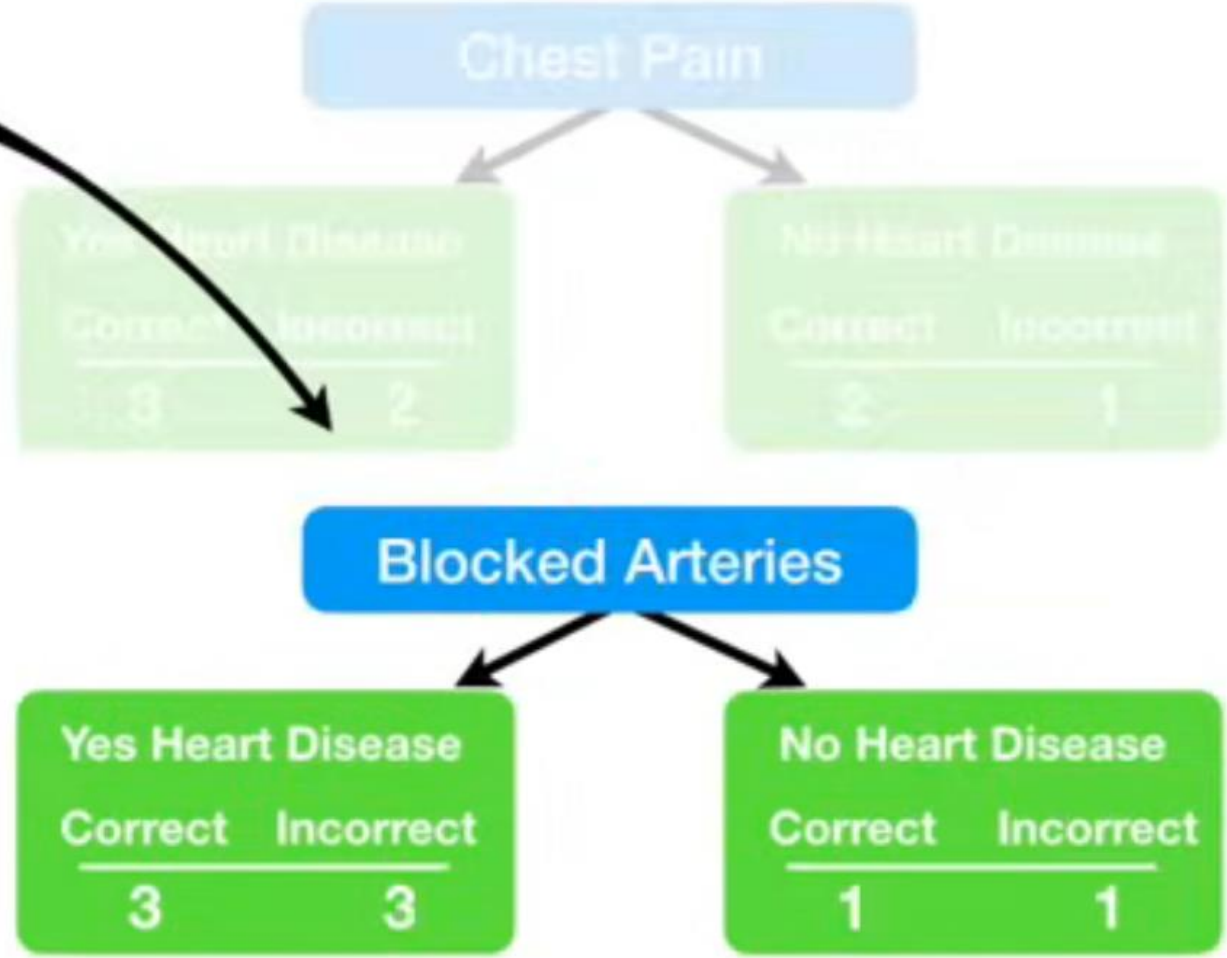
We start by seeing how well **Chest Pain** classifies the samples.

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



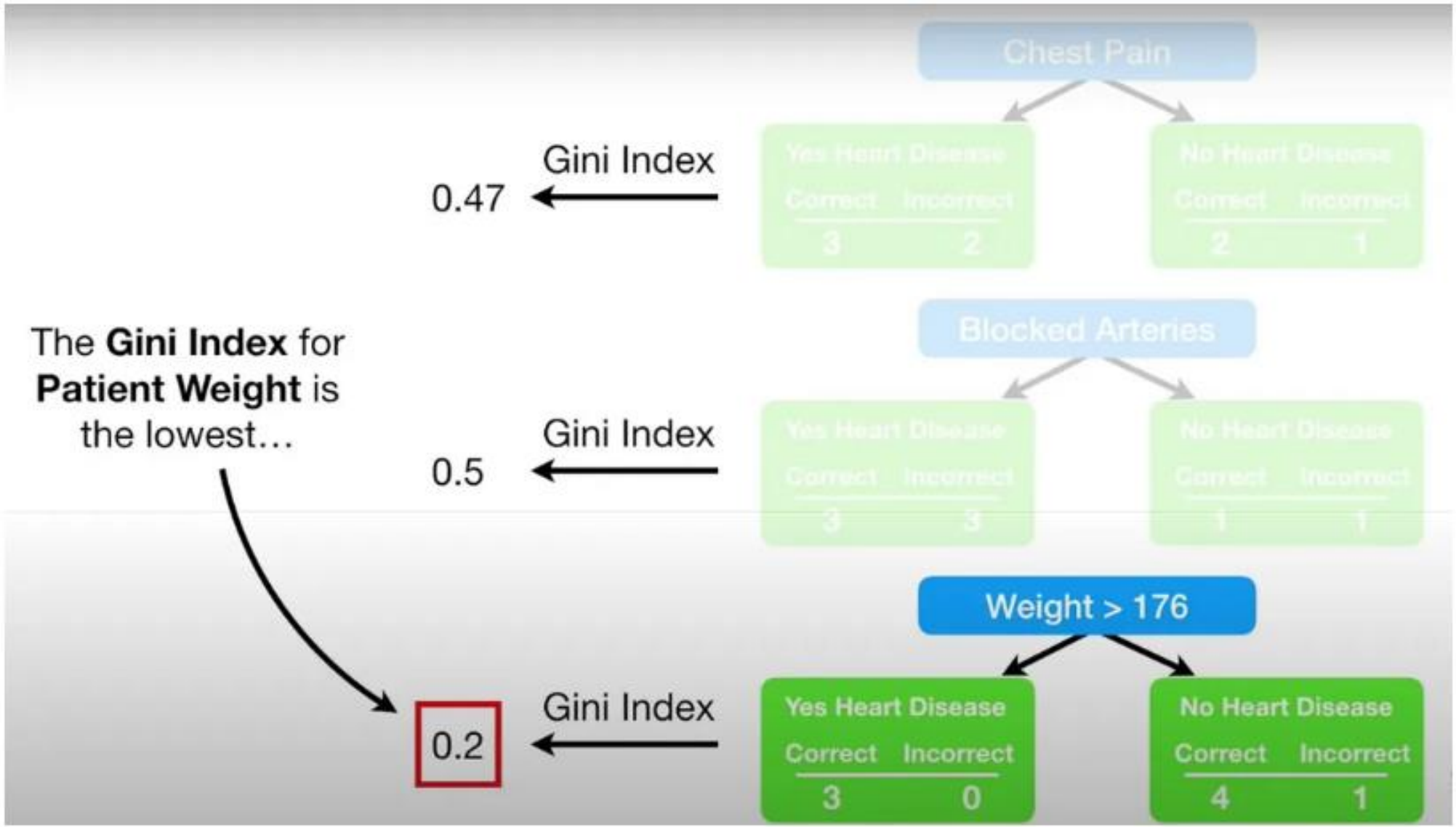
We start by seeing how well **Chest Pain** classifies the samples.

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



Now we do the same thing for **Blocked Arteries**…

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8


Chest Pain
Yes Heart Disease
Correct Incorrect
3 2
No Heart Disease
Correct Incorrect
2 1
Gini Index
0.47
Blocked Arteries
Yes Heart Disease
Correct Incorrect
3 3
No Heart Disease
Correct Incorrect
1 1
Gini Index
0.5
The Gini Index for Patient Weight is the lowest…
Weight > 176
Yes Heart Disease
Correct Incorrect
3 0
No Heart Disease
Correct Incorrect
4 1
Gini Index
0.2

…so this will be the first stump in the forest.

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.

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

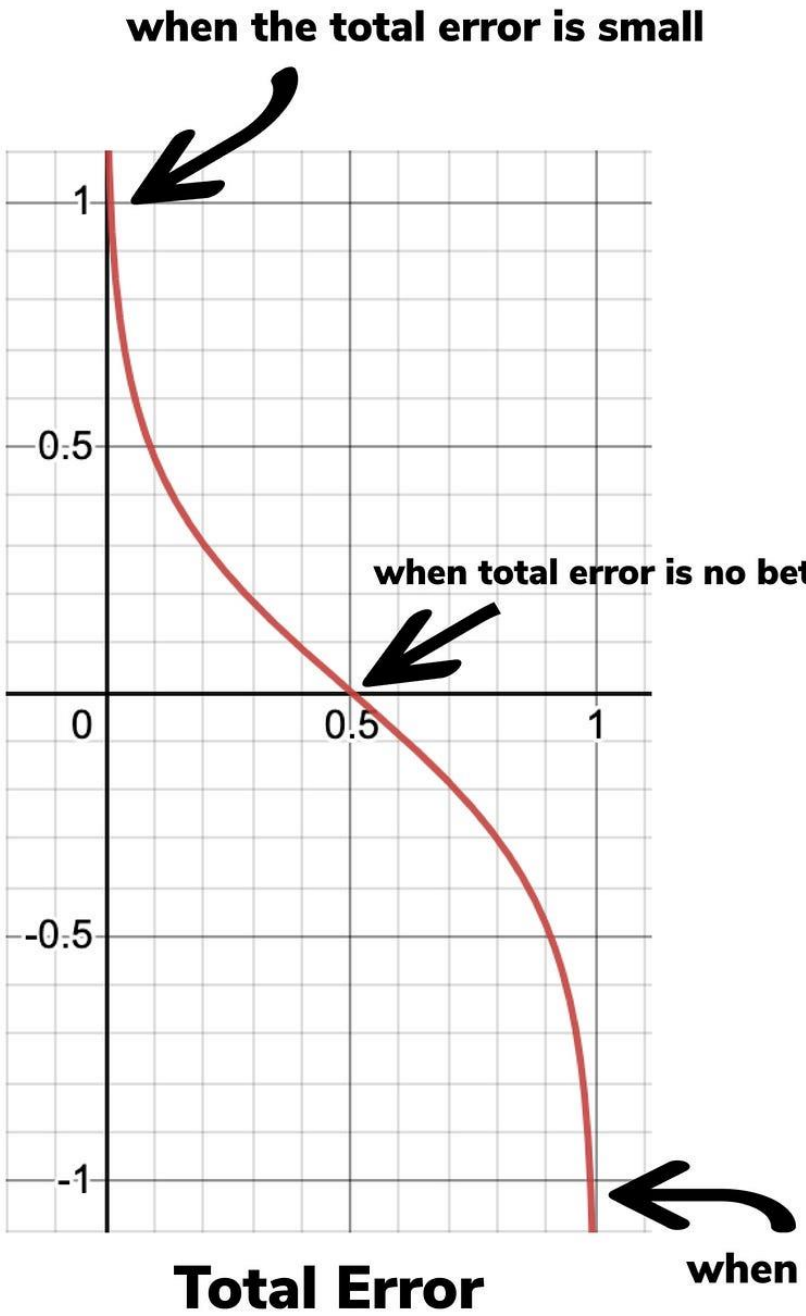
The **Total Error** for a stump is the sum of the weights associated with the *incorrectly* classified samples.

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

Thus, in this case, the **Total Error** is **1/8**.

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

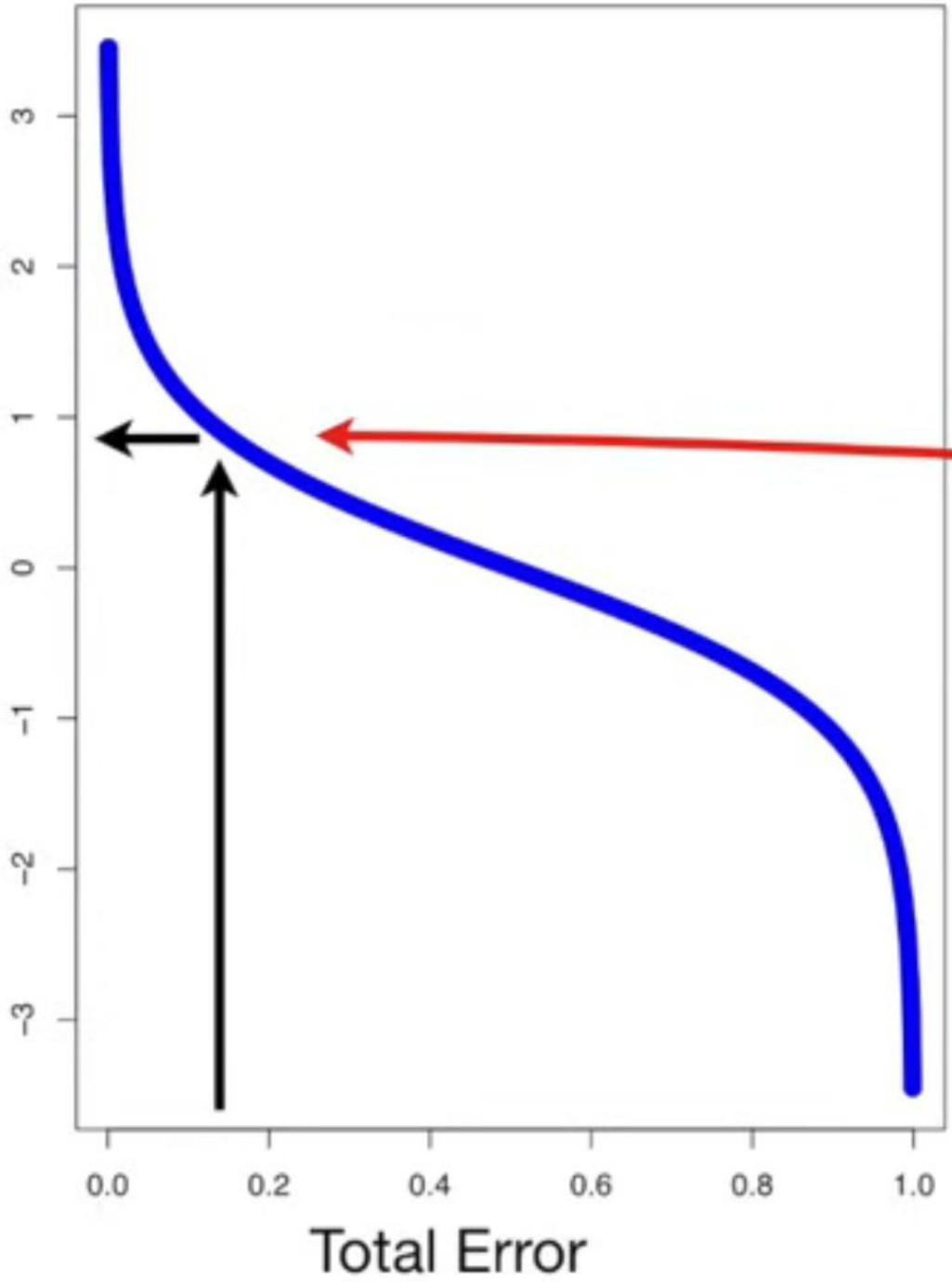
$$Amount\ of\ Say = \frac{1}{2}\ log(\frac{1 - Total\ Error}{Total\ Error})$$



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

The weighted error rate of the stump
(Amount of say of the stump)

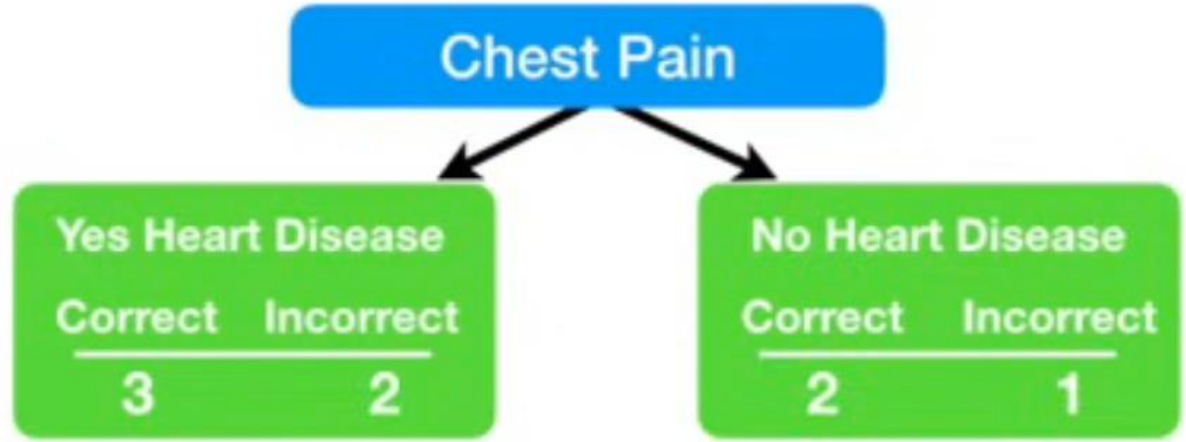
$$\alpha = \frac{1}{2}\ln\left(\frac{1-\epsilon}{\epsilon}\right)$$



…and the **Amount of Say** that this stump has on the final classification is **0.97**.

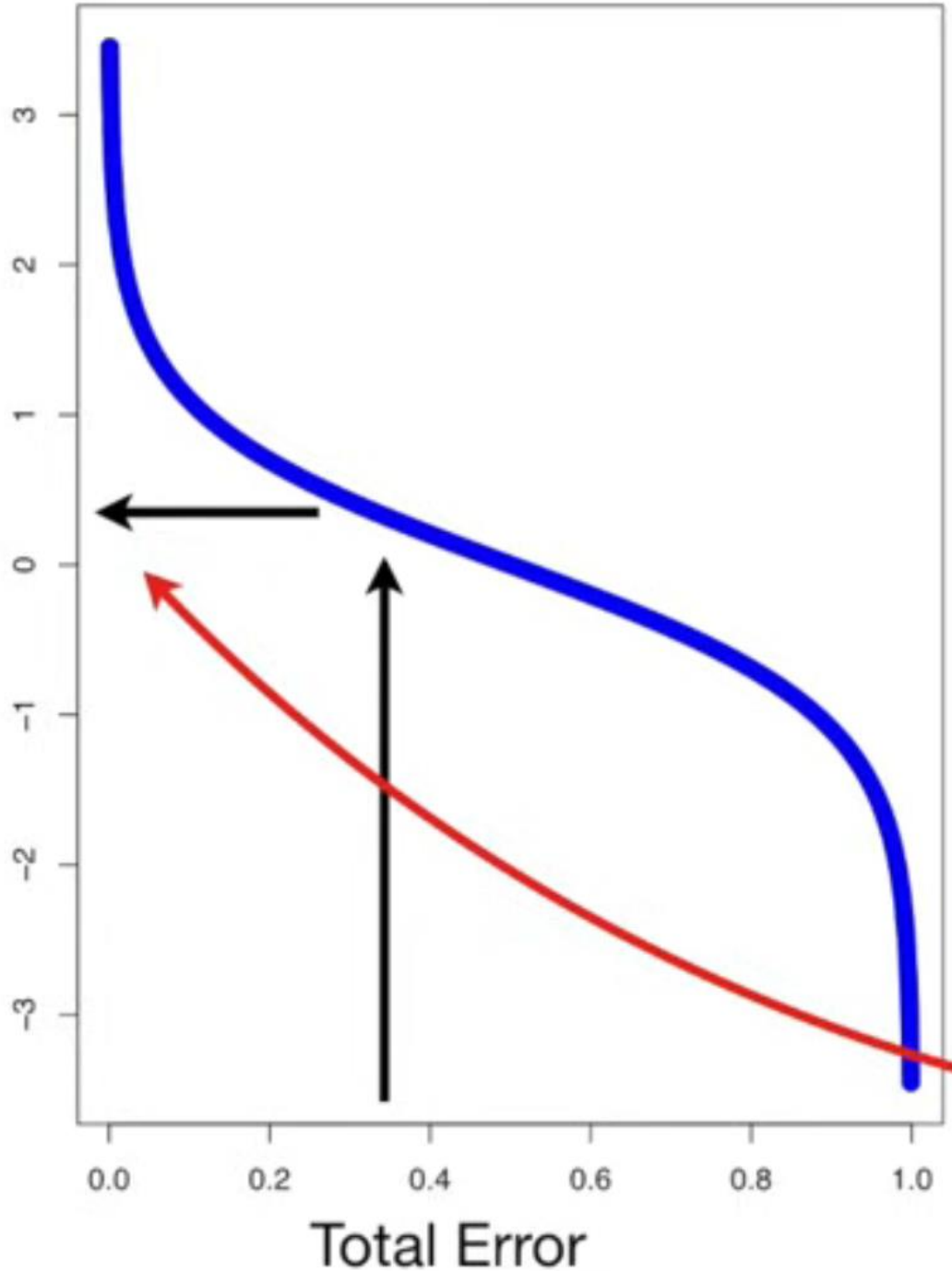
$$\text{Amount of Say} = \frac{1}{2}\log(7) = 0.97$$

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

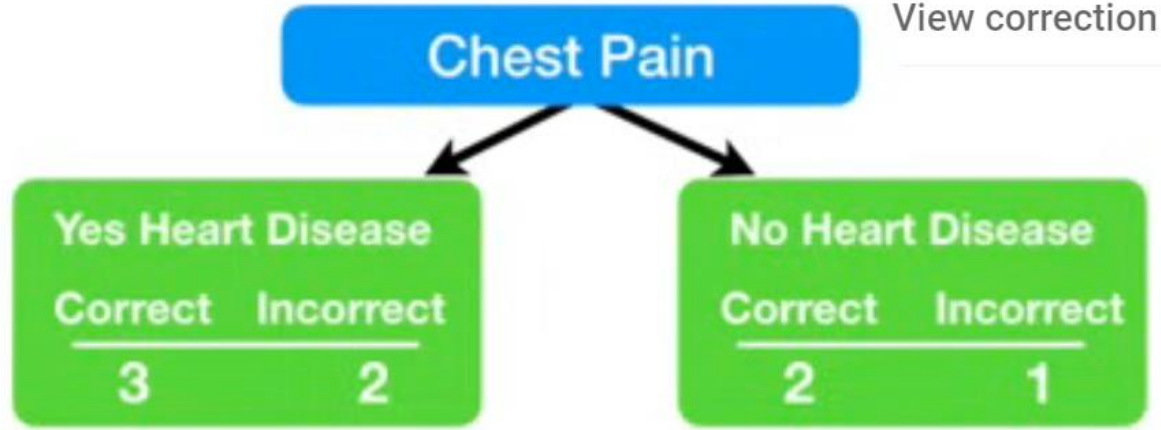


$$\text{Total Error} = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{3}{8}$$

So the **Total Error** for **Chest Pain** is **3/8**.



View correction



…and the **Amount of Say** that the **Chest Pain** stump would have had on the final classification is **0.42**.

$$\text{Amount of Say} = \frac{1}{2}\log(7/3) = 0.42$$

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

…we will emphasize the need for the next stump to correctly classify it by increasing its **Sample Weight**…

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
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

…and decreasing all of the other **Sample Weights**.

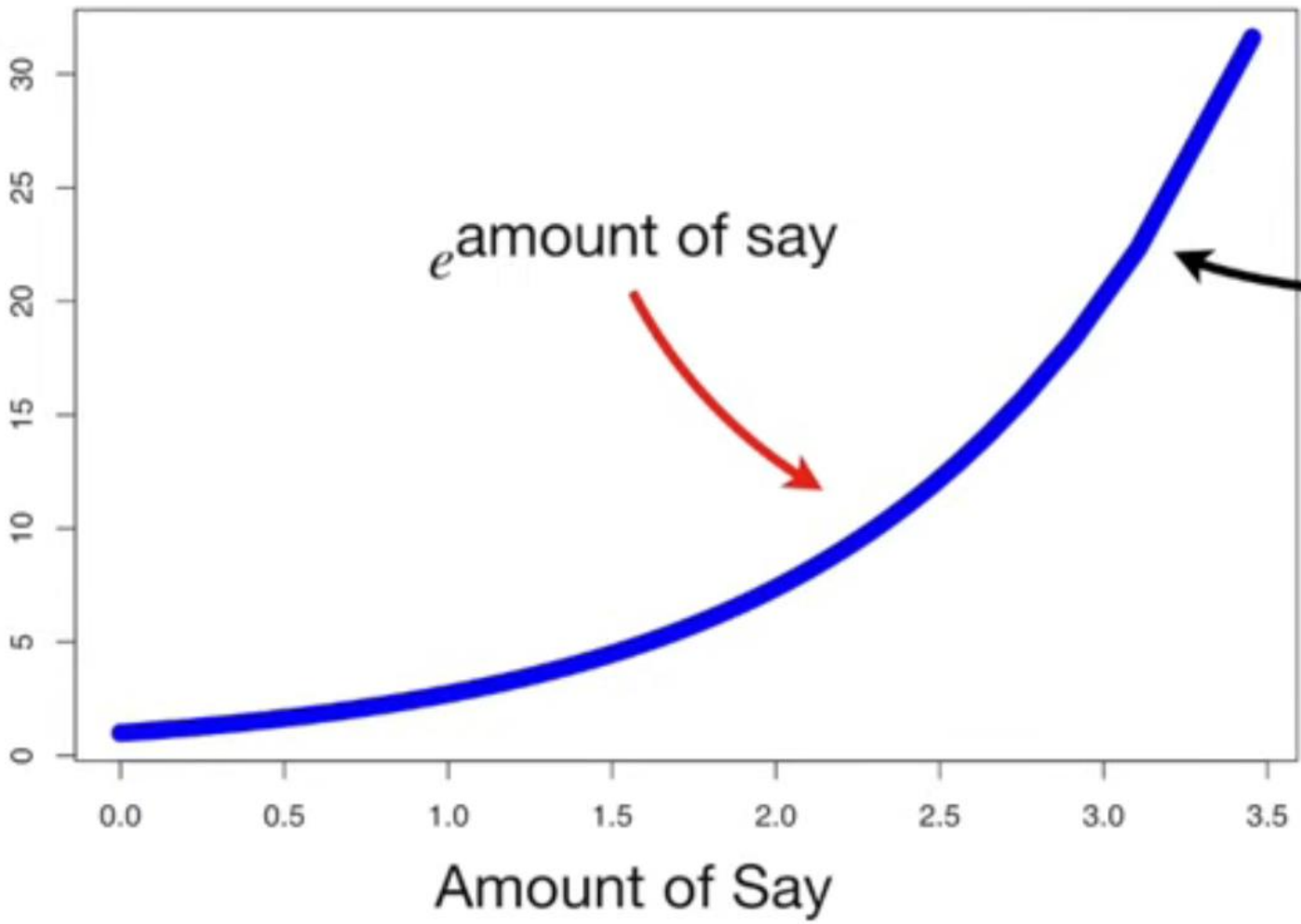
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

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

This is the formula we will use to *increase* the **Sample Weight** for the sample that was *incorrectly* classified.

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

$$= \frac{1}{8} e^{\text{amount of say}}$$



…then we will scale the previous **Sample Weight** with a large number.

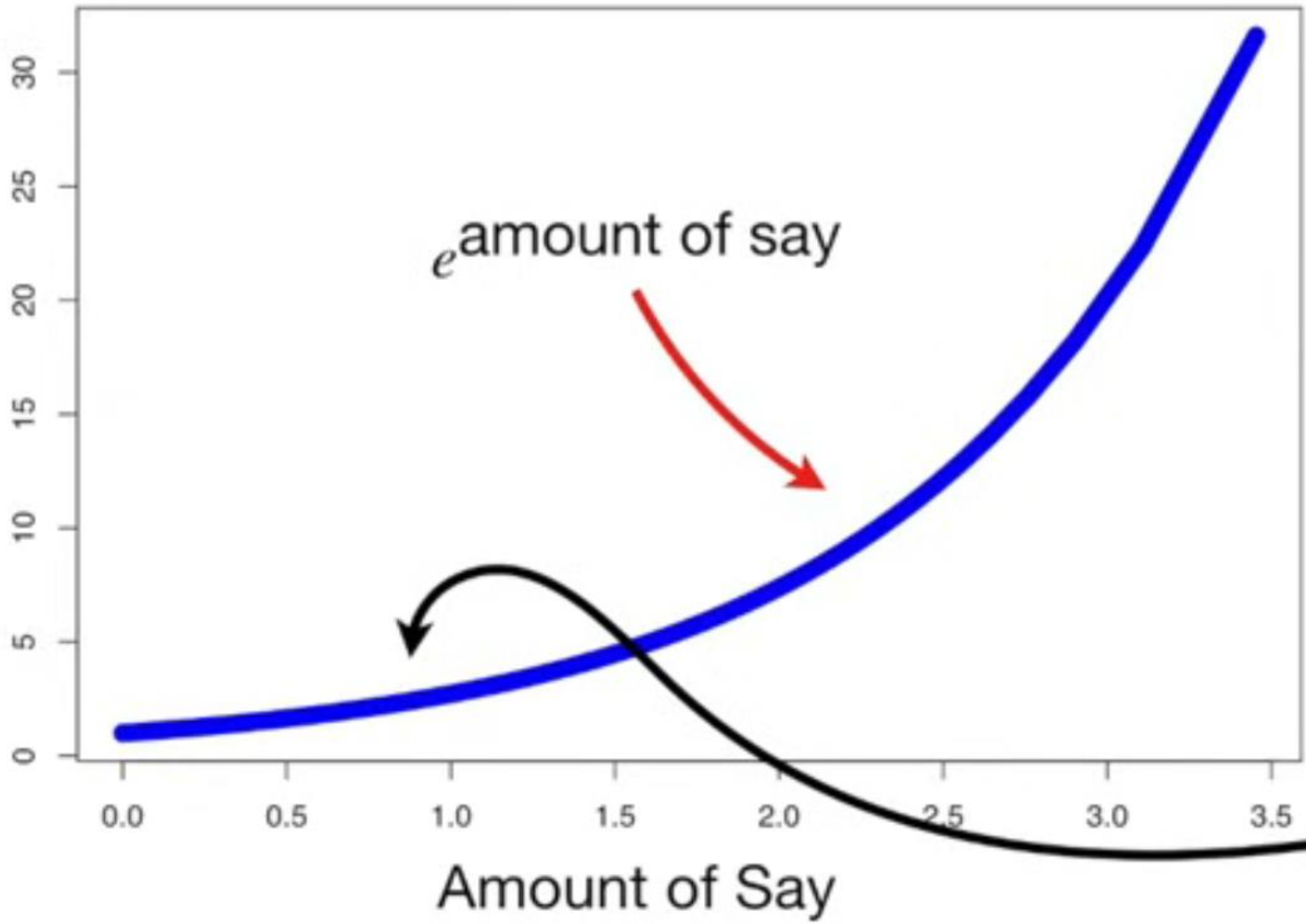
This means that the **New Sample Weight** will be much larger than the old one.

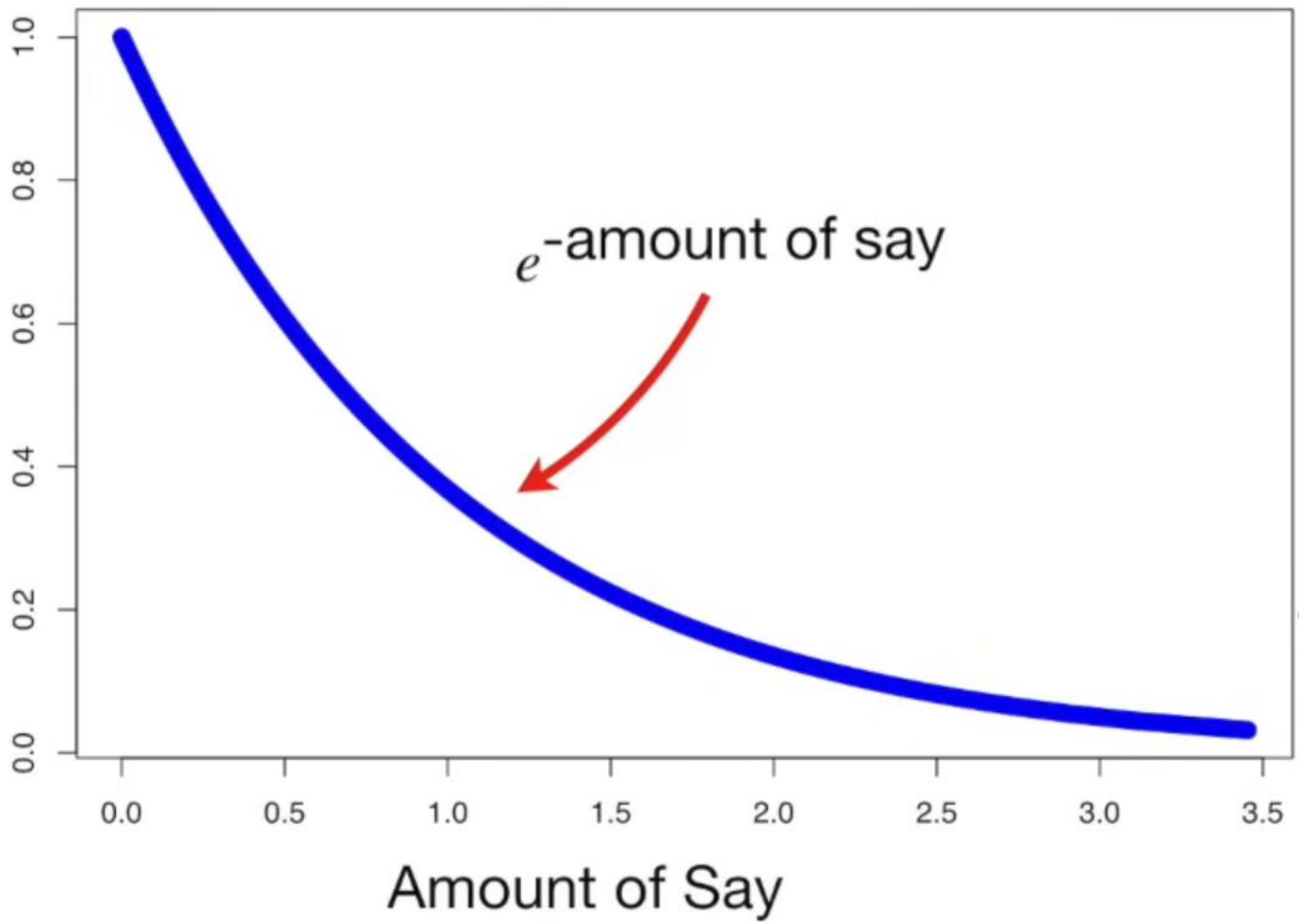
New Sample Weight = sample weight $\times e^{\text{amount of say}}$

$= \frac{1}{8} e^{\text{amount of say}}$

$= \frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64$

…and $e^{0.97} = 2.64$





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

$$= \frac{1}{8} e^{\text{-amount of say}}$$

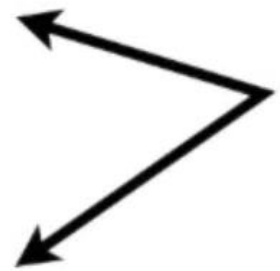
$$= \frac{1}{8} e^{-0.97} = \frac{1}{8} \times 0.38 = 0.05$$

The new **Sample Weight** is **0.05**, which is *less* than the old one (**1/8** = **0.125**).

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New 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	0.33
No	Yes	156	No	1/8	
No	Yes	125	No	1/8	
Yes	No	168	No	1/8	
Yes	Yes	172	No	1/8	

We plug in **0.33** for the sample that was *incorrectly* classified…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05



All of the other samples get **0.05**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

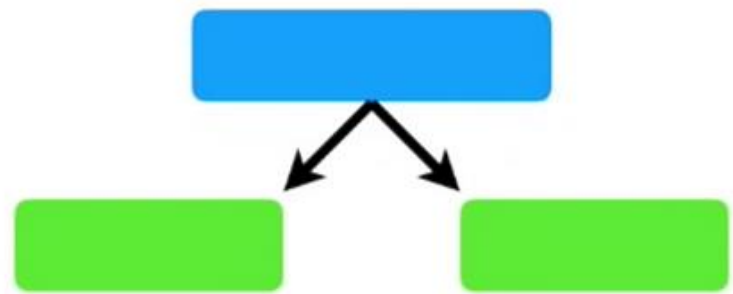
Right now, if you add up the **New Sample Weights**, you get **0.68**.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

So we divide each **New Sample Weight** by **0.68** to get the normalized values.

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

Now we can use the modified **Sample Weights** to make the second **stump** in the forest.



Two options

- Weighted Gini index for creating a new stump

- Generating a new dataset (same size as the original ONE) containing duplicate copies of sample points (data points) which are having large weights

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	
Yes	Yes	167	Yes	
No	Yes	156	No	
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

So we start by making a new, but empty, dataset that is the same size as the original…

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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

We then continue to pick random numbers and add samples to the new collection until we the new collection is the same size as the original.

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

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	Yes	0.07
Yes	Yes	172	No	0.07

Ultimately, this sample was added to the new collection of samples **4** times, reflecting its larger **Sample Weight**.

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

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

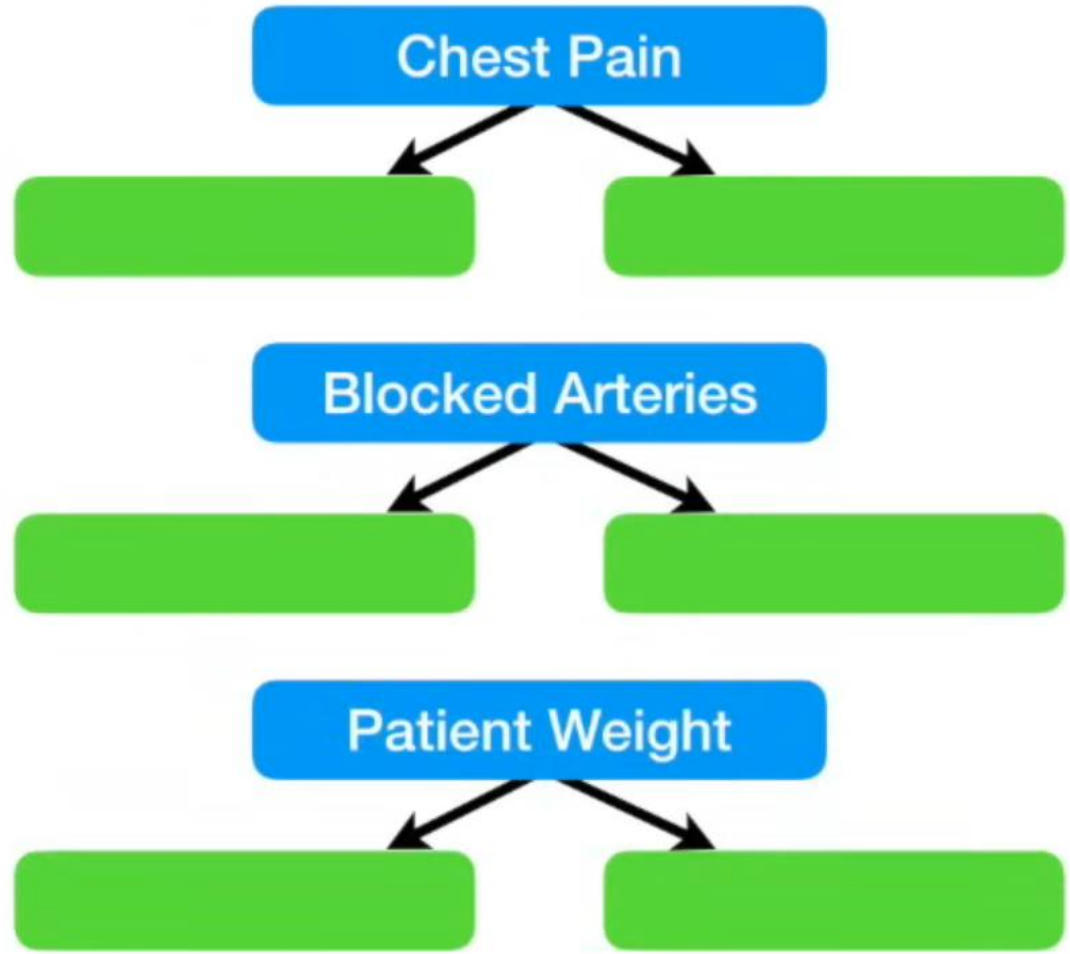
Lastly, we give all the samples equal **Sample Weights**, just like before.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

Because these samples are all the same, they will be treated as a block, creating a large penalty for being misclassified.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

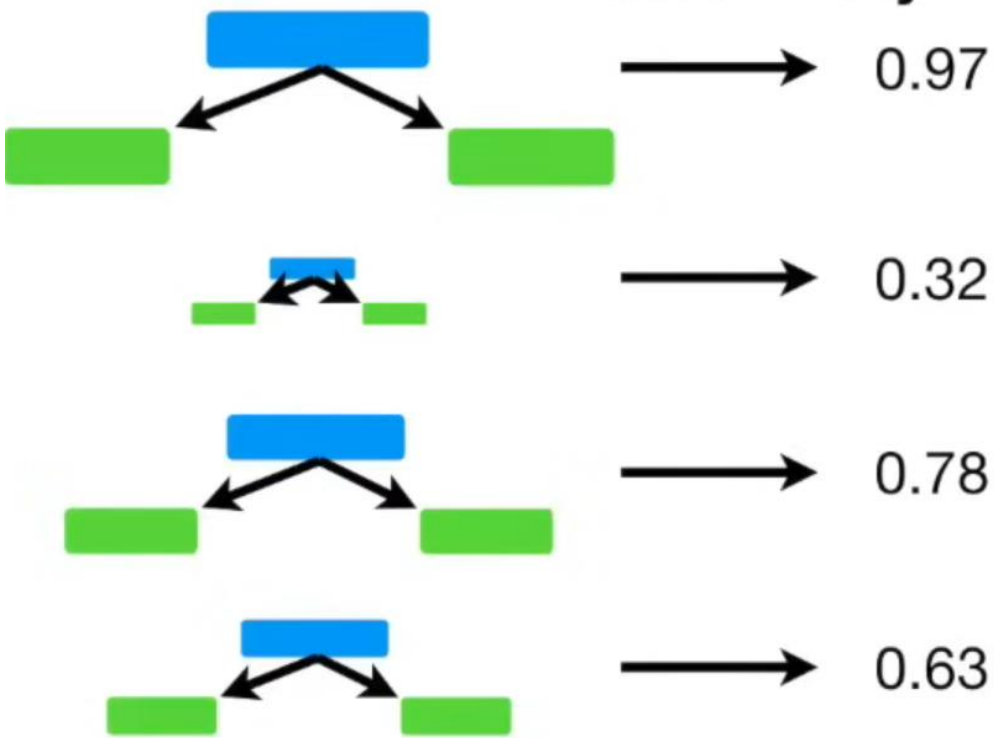
Now we go back to the beginning and try to find the stump that does the best job classifying the new collection of samples.



Now we add up the **Amounts of Say** for this group of stumps…

Has Heart Disease **Total = 2.7**

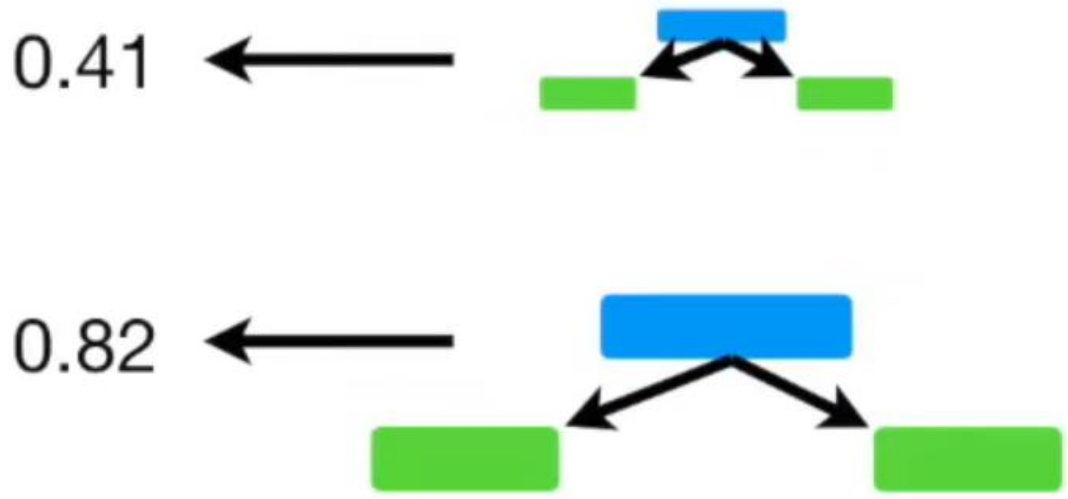
Amount of Say



…and for this group of stumps…

Total = 1.23 Does Not Have Heart Disease

Amount of Say



Ultimately, the patient is classified as **Has Heart Disease** because this is the larger sum.

Has Heart Disease

Total = 2.7

Total = 1.23

Does Not Have Heart Disease

Amount of Say

0.97

0.32

0.78

0.63

Amount of Say

0.41

0.82

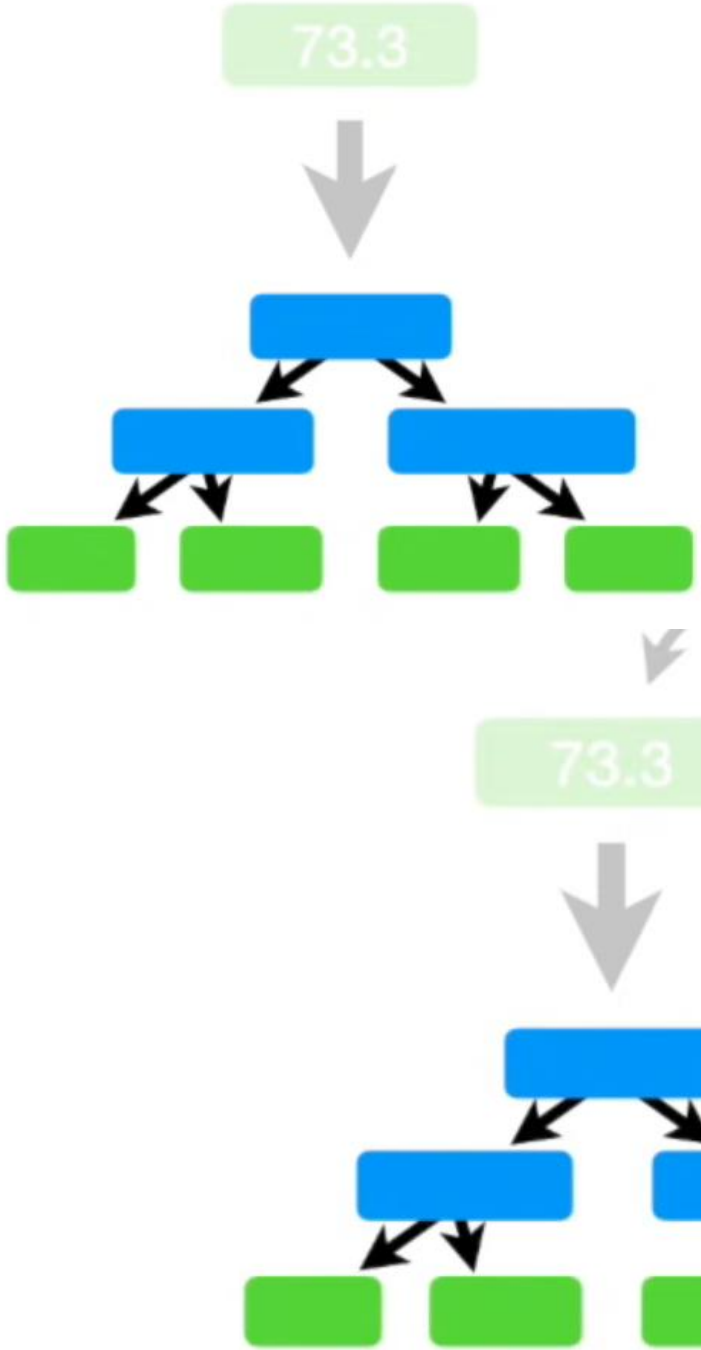
Summary

Let's summarize the pros and cons of AdaBoost as compared to other supervised machine learning models:

- **Pros:**
- Can provide highly accurate models, as each weak learner improves the areas where previous models performed poorly.
- Has strong theoretical guarantees regarding the training error (the error decreases exponentially in the number of boosting rounds).
- Less prone to overfitting, since it is built from an ensemble of weak learners, where each learner underfits the training set.
- **Cons:**
- Sensitive to outliers and mislabeled data points, since examples that are difficult to classify are assigned ever-increasing weights in successive boosting rounds.
- Training can be computationally intensive when dealing with large data sets

Gradient boosting

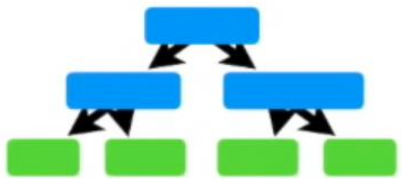
- Gradient boosting is a widely used machine learning technique that is based on a combination of **boosting** and **gradient descent**.
- Like AdaBoost, it combines multiple weak learners (or base learners) to create a strong predictive model. The base models are trained sequentially, where each model focuses on correcting the errors made by the previous models.
- When the weak learners are decision trees, the resulting method is known as **gradient-boosted decision trees** (GBDT) or **gradient boosting machine** (GBM).
- Variants → XGBoost, CatBoost, LightGBM



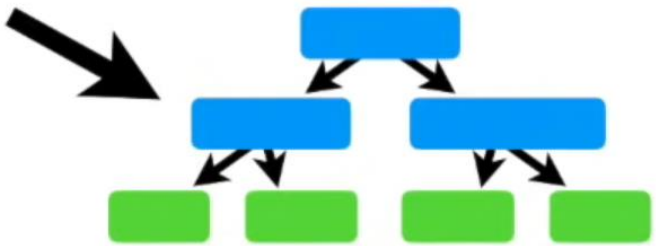
Thus, like **AdaBoost**, **Gradient Boost** builds fixed sized trees based on the previous tree's errors, but unlike **AdaBoost**, each tree can be larger than a stump.

Also like **AdaBoost**, **Gradient Boost** scales the trees. However, **Gradient Boost** scales all trees by the same amount.

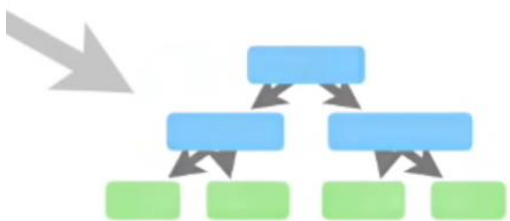
73.3

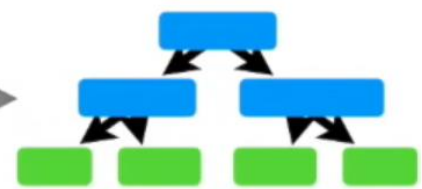


Then **Gradient Boost** builds another tree based on the errors made by the previous tree…



…and **Gradient Boost** continues to build trees in this fashion until it has made the number of trees you asked for, or additional trees fail to improve the fit.





etc. etc. etc.

…let's see how the most common **Gradient Boost** configuration would use this **Training Data** to **Predict Weight**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Average Weight

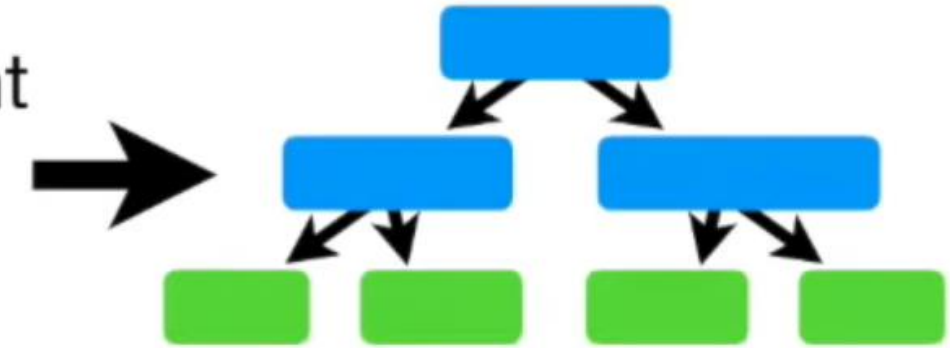
71.2

In other words, if we stopped right now, we would predict that everyone **Weighed 71.2** kg.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Average Weight

71.2

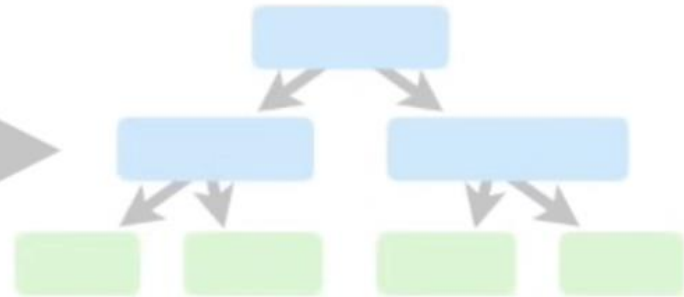


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56

The next thing we do is build a tree based on the errors from the first tree.

Average Weight

71.2



The errors that the previous tree made are the differences between the **Observed Weights** and the **Predicted Weight**, **71.2**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

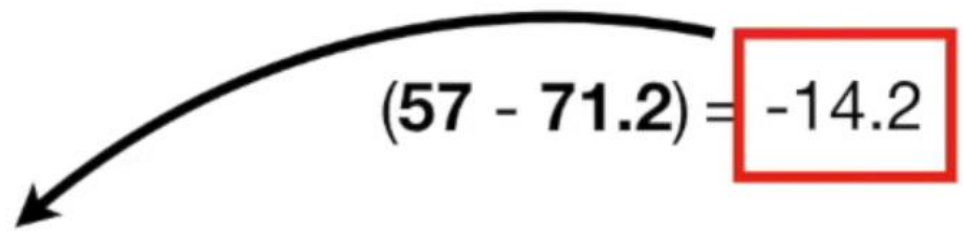
(Observed Weight - Predicted Weight)

Average Weight

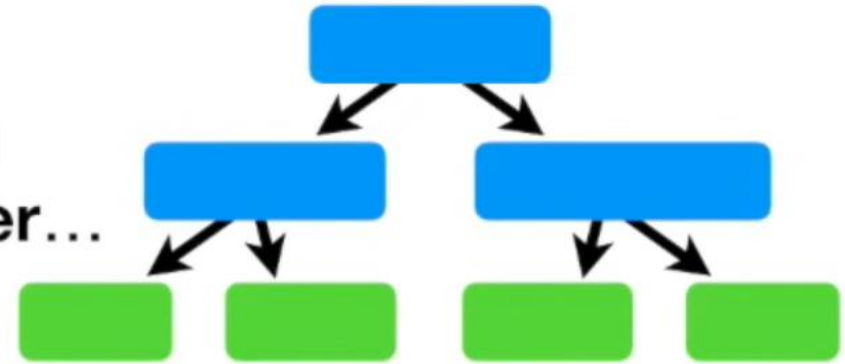
71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

Now do the same thing for the remaining **Weights**…



Now we will build a **Tree**, using **Height**, **Favorite Color** and **Gender**…



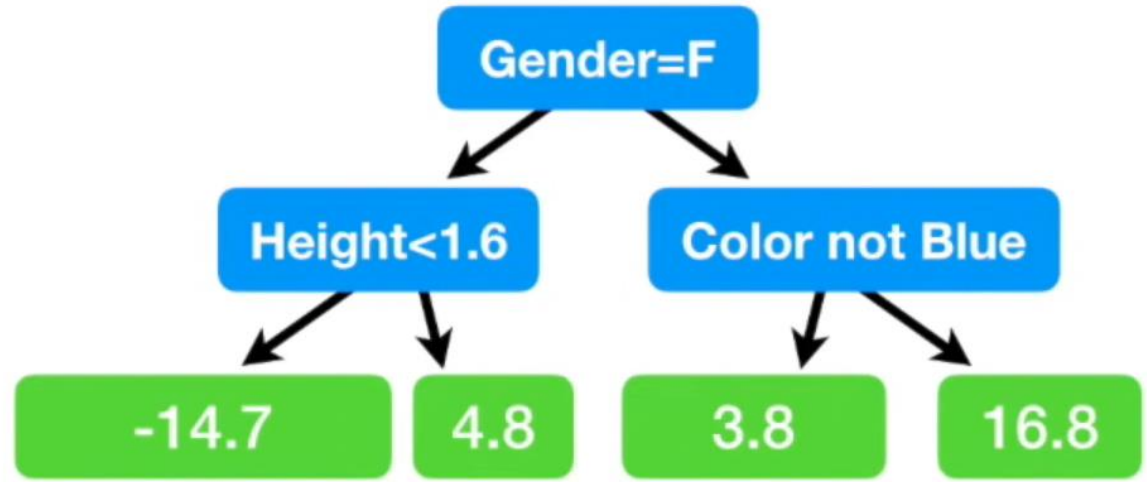
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

…to **Predict** the **Residuals**.

Average Weight

71.2

+



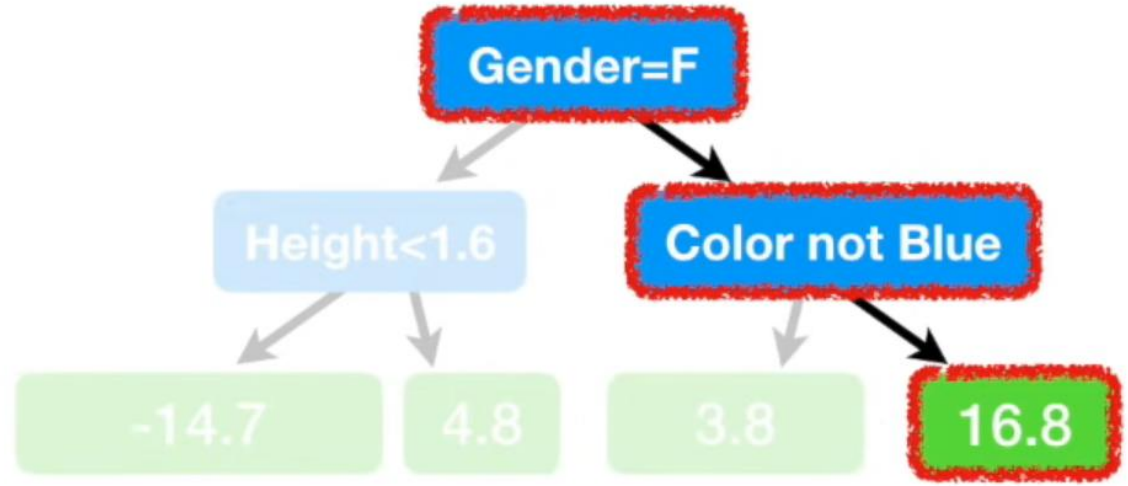
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

…to make a new **Prediction** of an individual's **Weight** from the **Training Data**.

Average Weight

71.2

+



…so the **Predicted Weight** = 71.2 + 16.8 = 88

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

Predicted Weight = 71.2 + 16.8 = 88

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

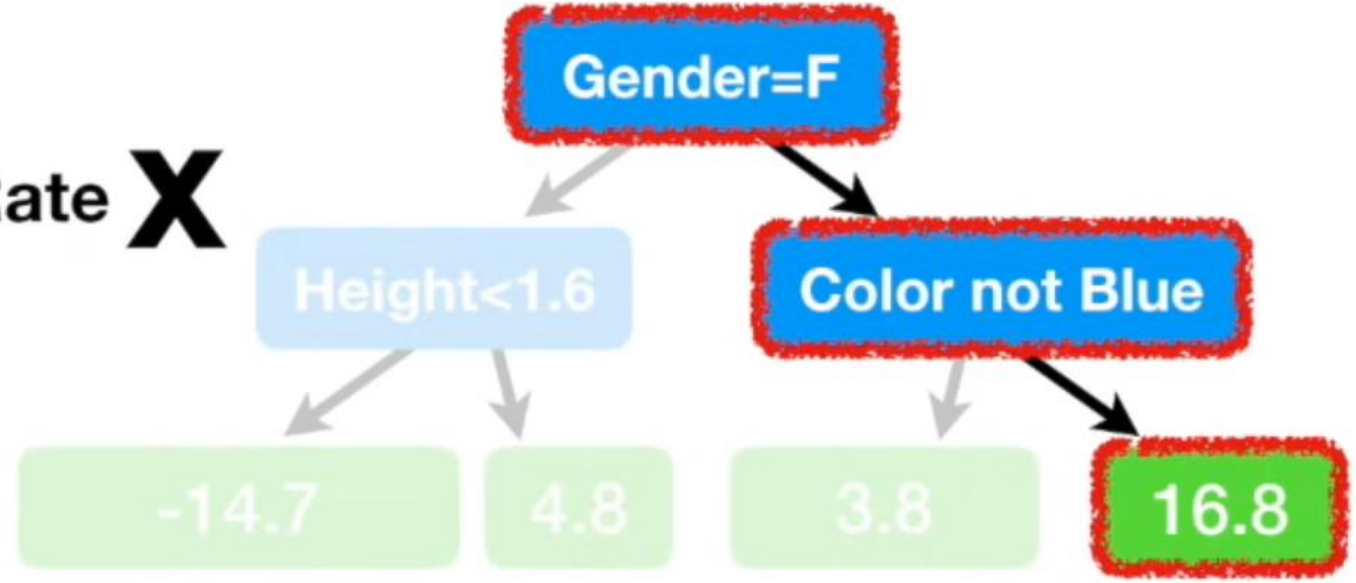
No. The model fits the **Training Data** too well.

In other words, we have low **Bias**, but probably very high **Variance**.

Average Weight

71.2

+ Learning Rate X



Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

Gradient Boost deals with this problem by using a **Learning Rate** to scale the contribution from the new tree.

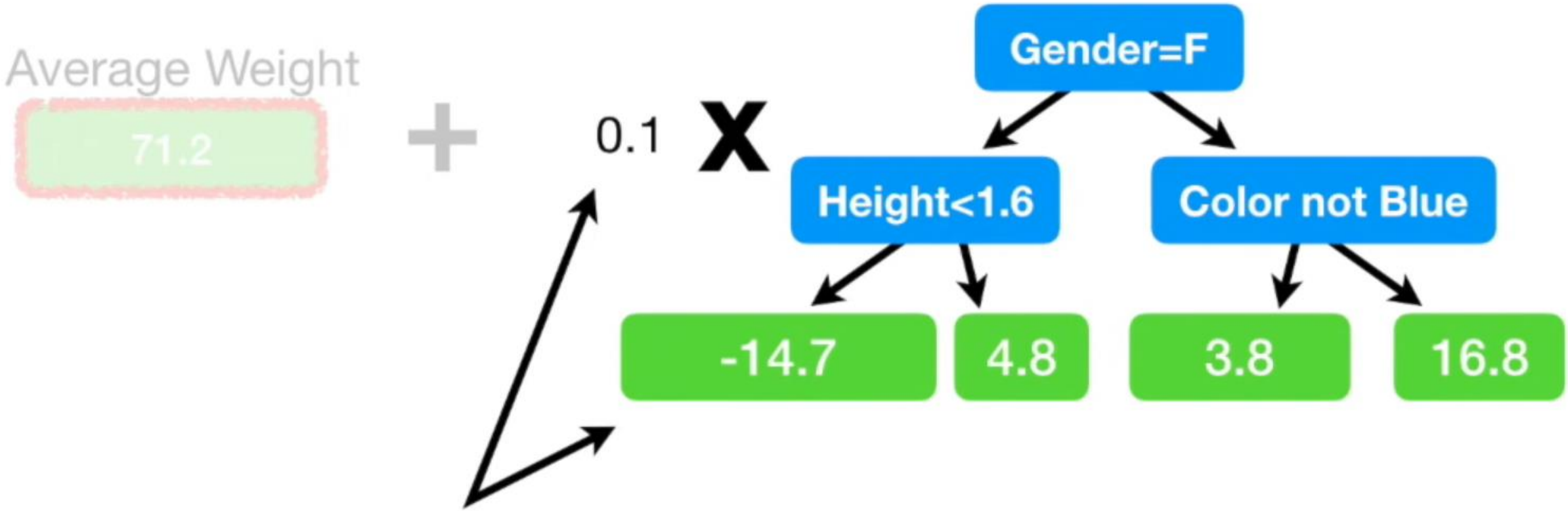
The **Learning Rate** is a value between **0** and **1**.

16.8

Predicted Weight = 71.2 + (0.1 × 16.8) = 72.9

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

With the **Learning Rate** set to **0.1**, the new **Prediction** isn't as good as as it was before…



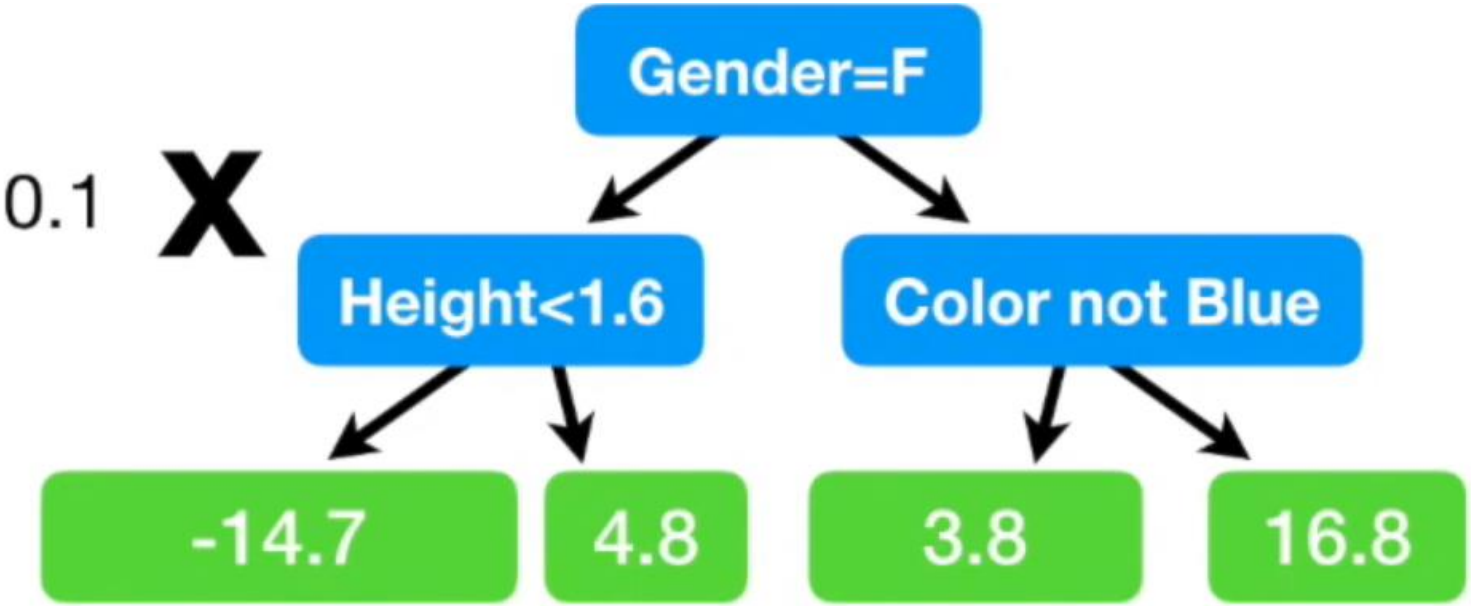
In other words, scaling the tree by the **Learning Rate** results in a small step in the right direction.

Average Weight

71.2

+

0.1 X



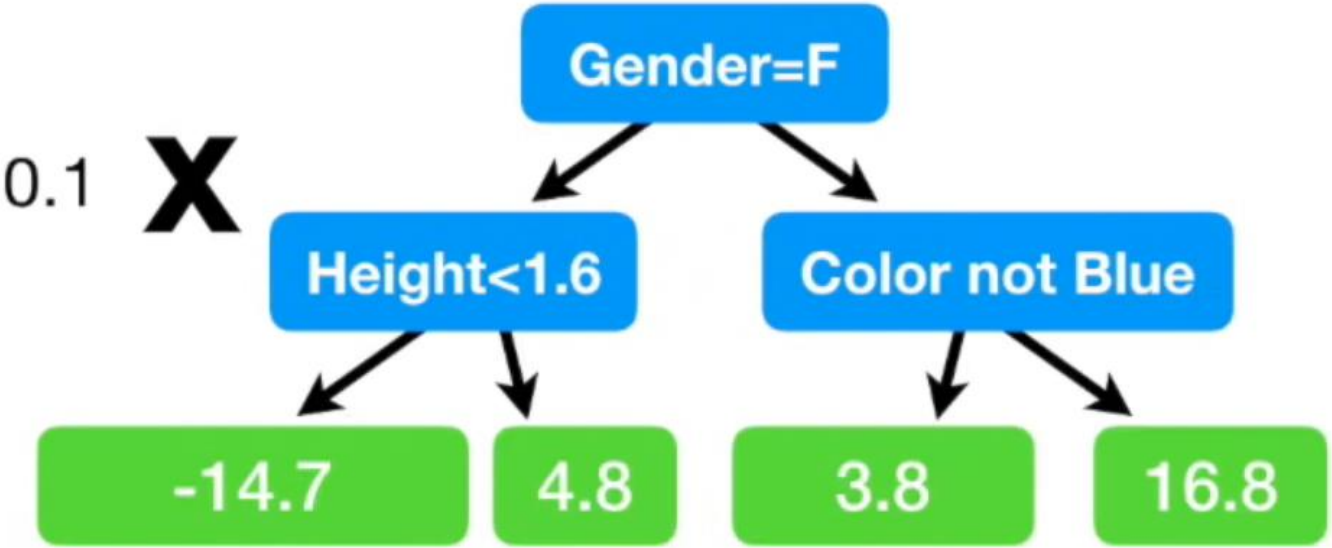
So let's build another tree so we can take another small step in the right direction.

Average Weight

71.2

+

0.1 X



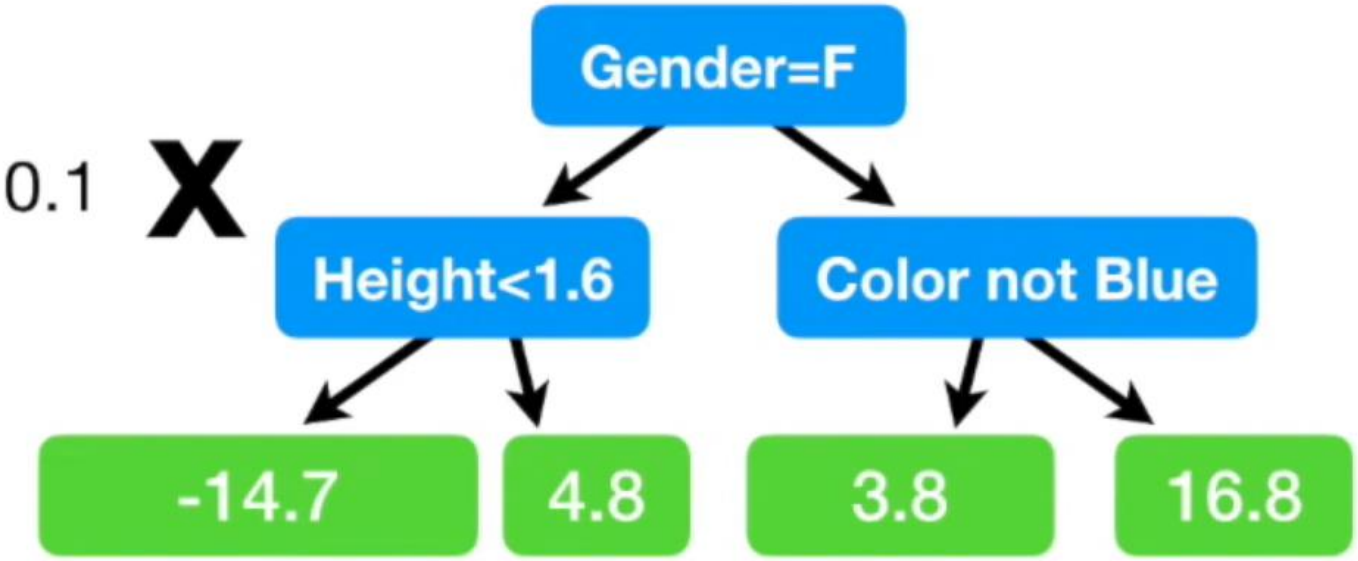
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

Average Weight

71.2

+

0.1 X

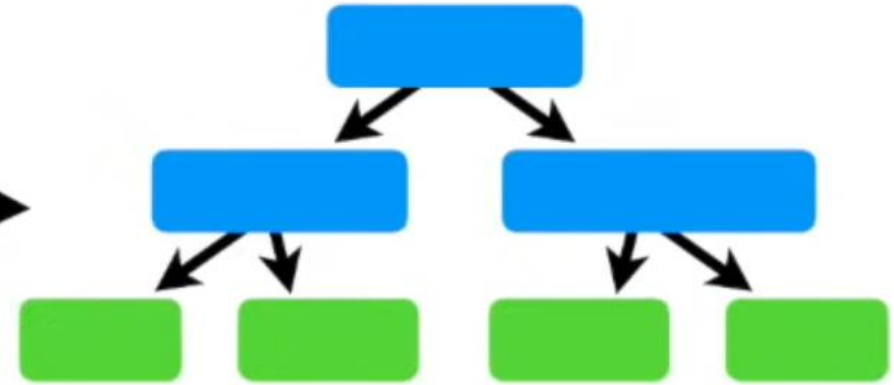


Residual
16.8
4.8
-15.2
1.8
5.8
-14.2

Residual
15.1
4.3
-13.7
1.4
5.4
-12.7

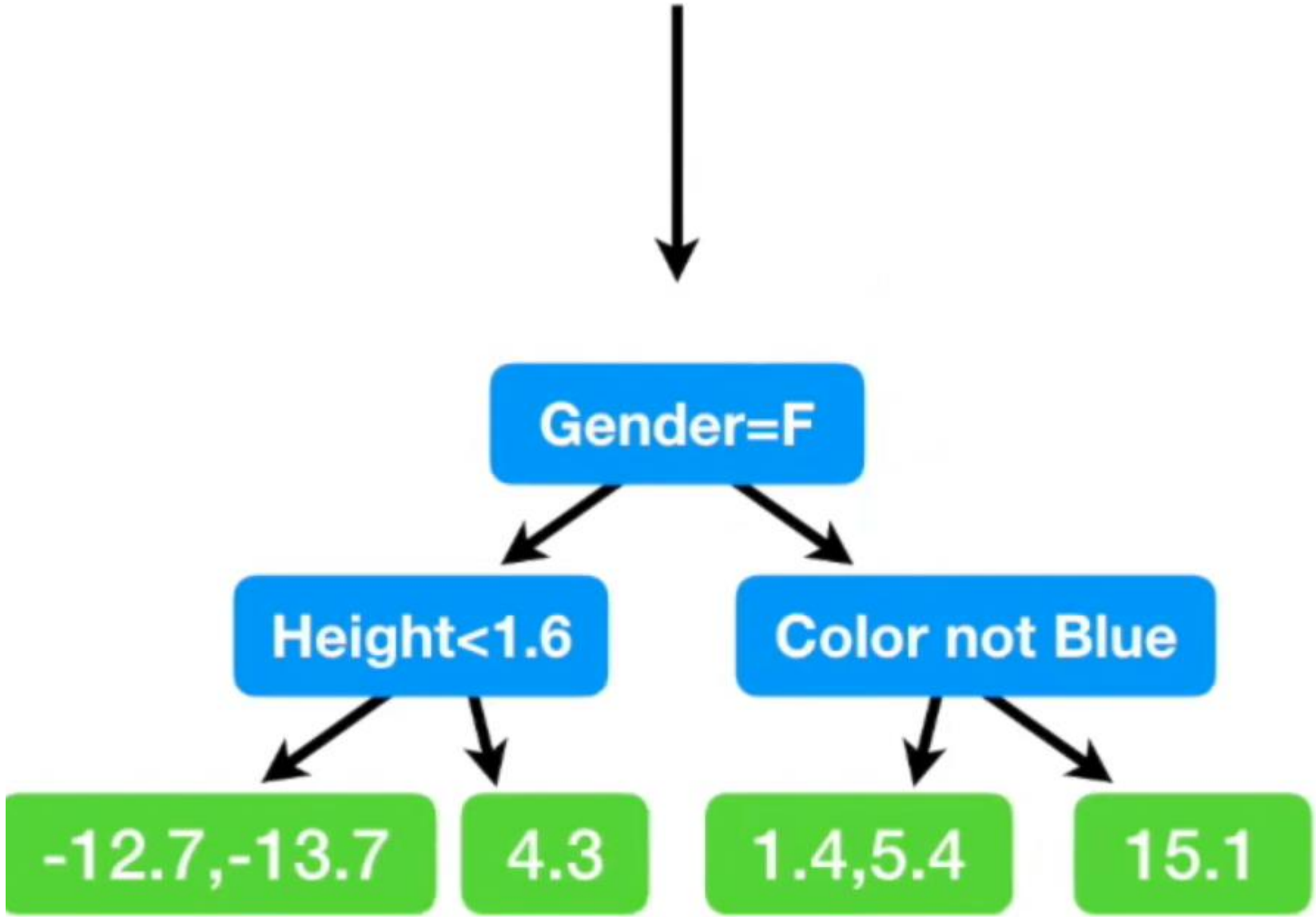
The new **Residuals** are all smaller than before, so we've taken a small step in the right direction.

Now let's build a new tree to predict the new **Residuals**.



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

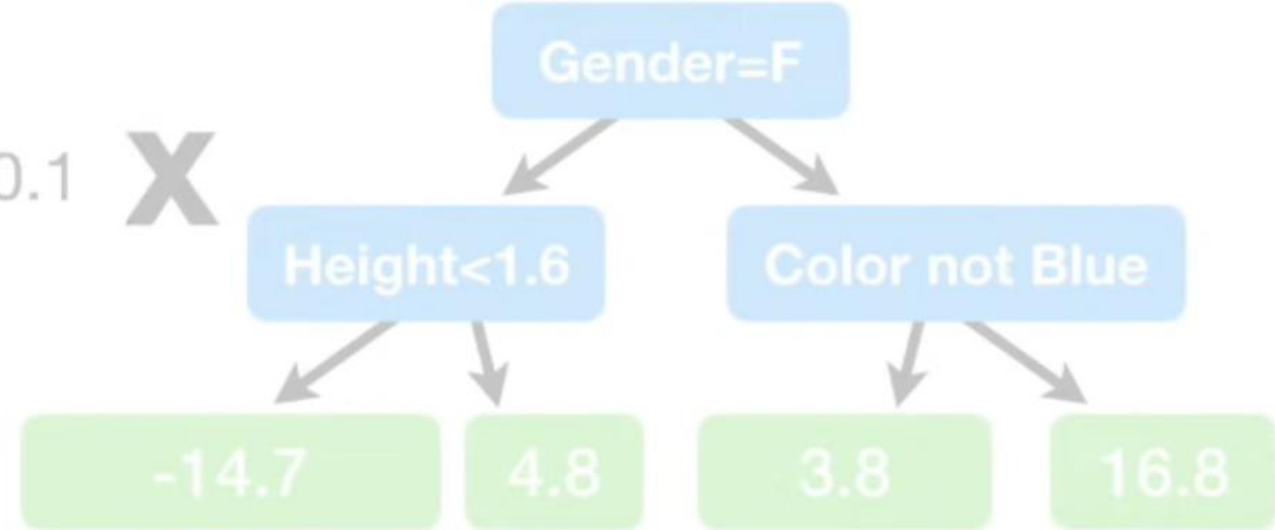
And here's the new tree!



Average Weight

71.2

+ 0.1 X



Now we're ready to make a new **Prediction** from the **Training Data**.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

+ 0.1 X

Average Weight

71.2

+ 0.1 X

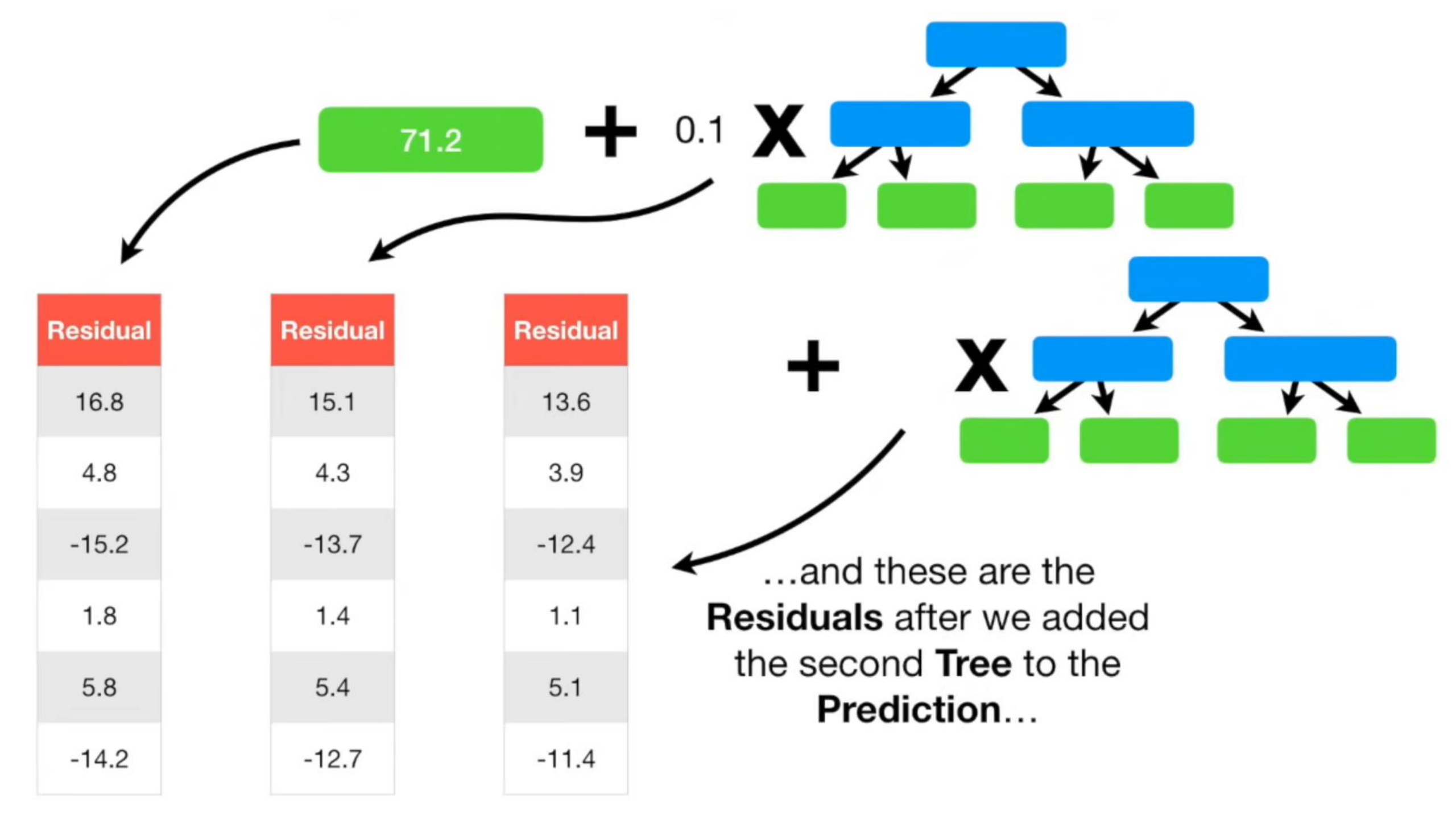
Which is another small step closer to the **Observed Weight**.

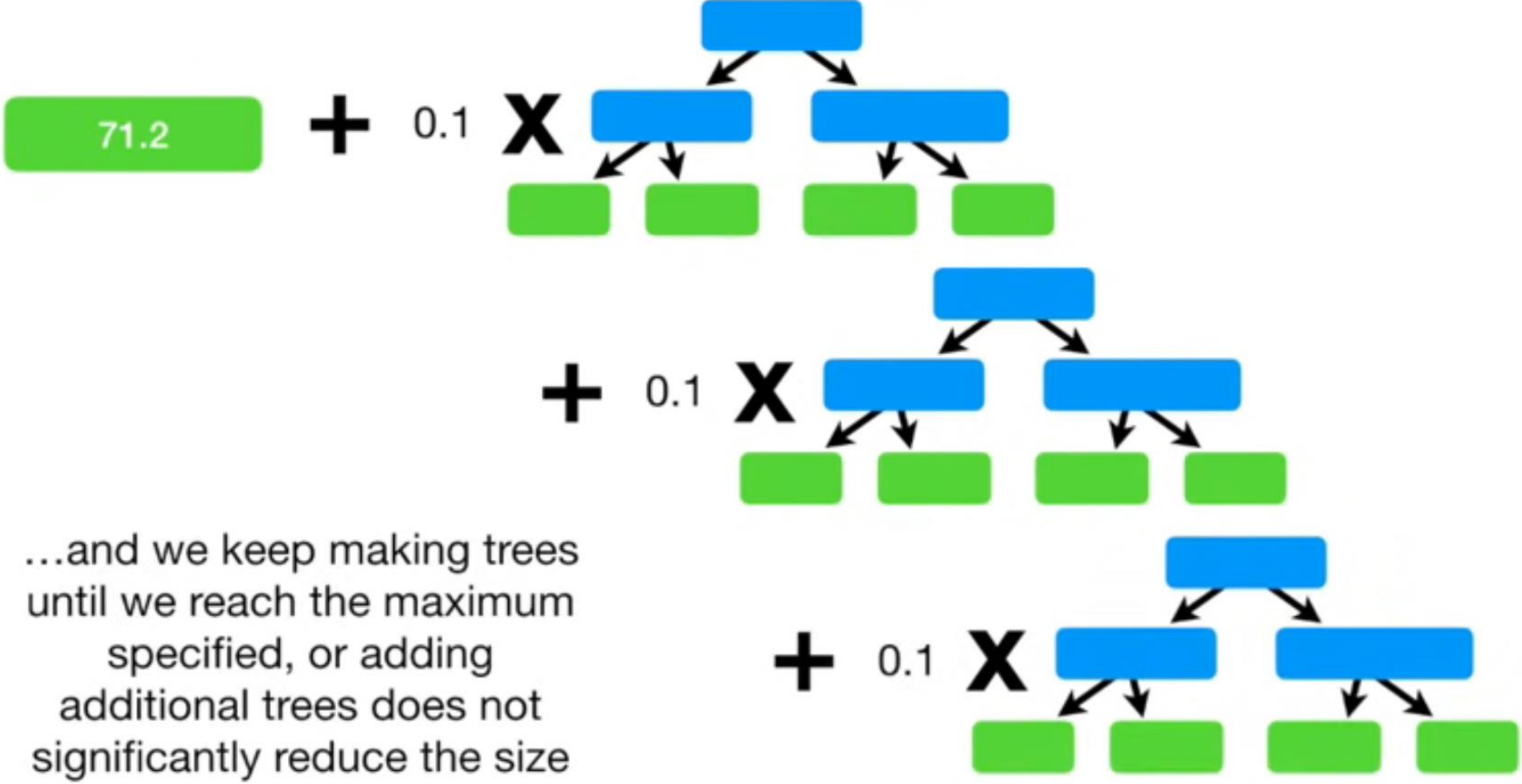
$71.2 + (0.1 \times 16.8) + (0.1 \times 15.1)$

= 74.4

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

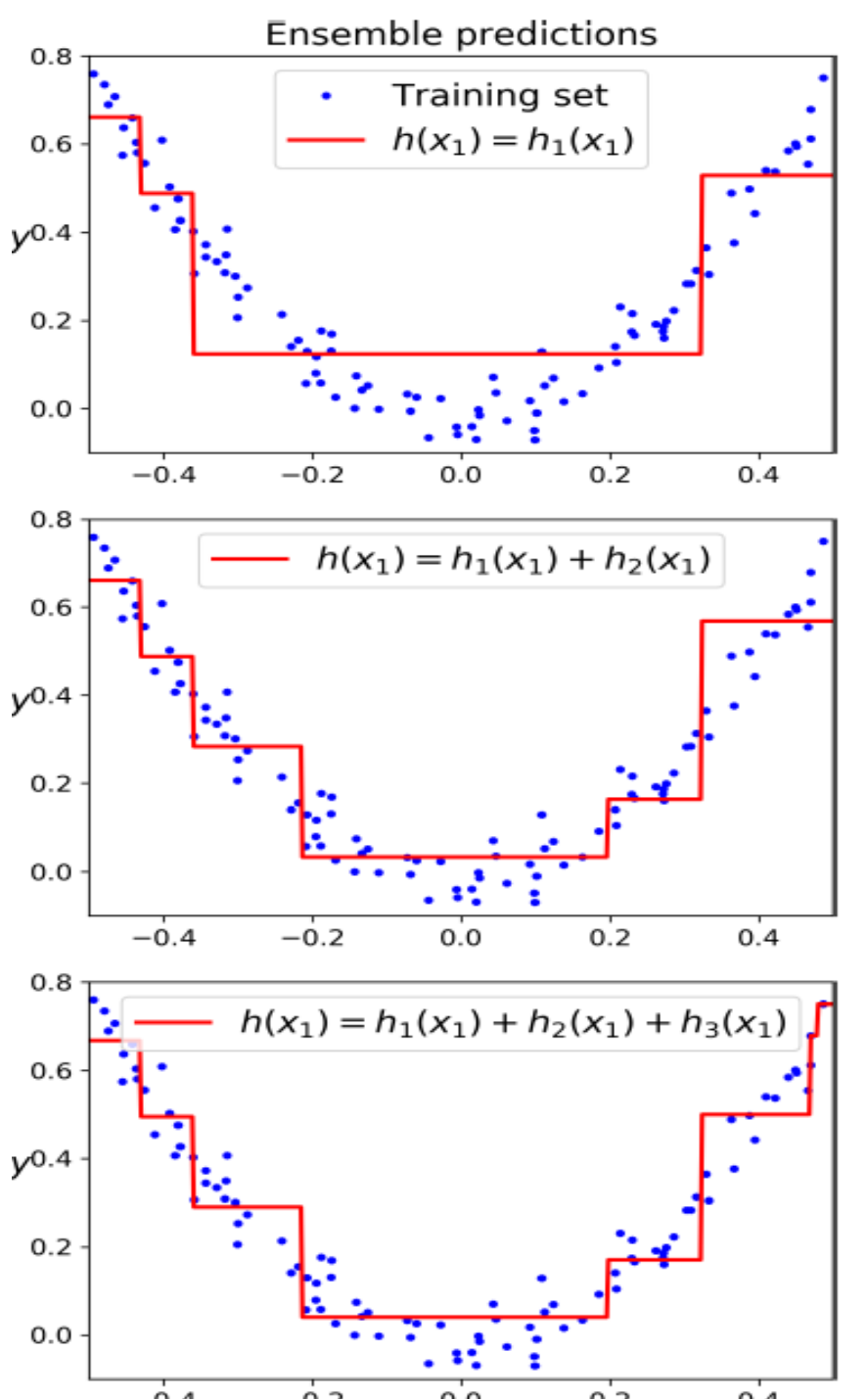
+ 0.1 X

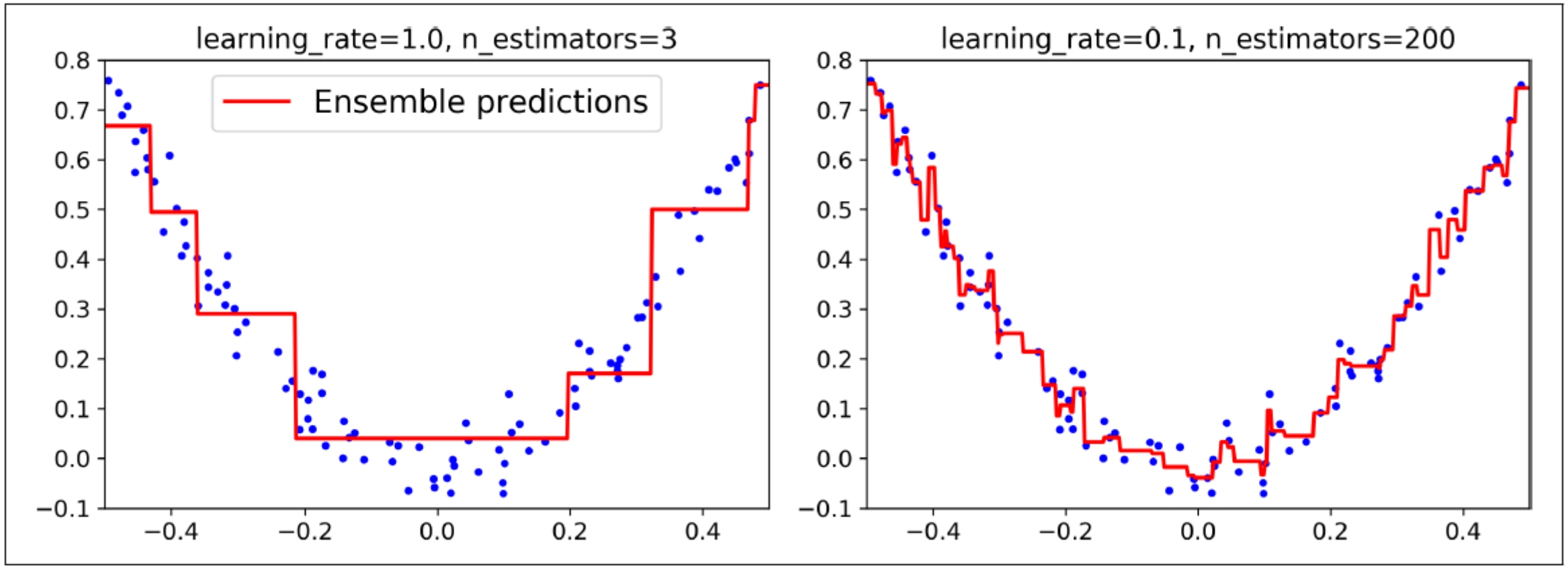

71.2
+
0.1
X
+
X
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7
Residual
13.6
3.9
-12.4
1.1
5.1
-11.4
…and these are the **Residuals** after we added the second **Tree** to the **Prediction**…



…and we keep making trees until we reach the maximum specified, or adding additional trees does not significantly reduce the size of the **Residuals**.

…etc…etc…etc…


Ensemble predictions
Training set
$h(x_1) = h_1(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4


learning_rate=1.0, n_estimators=3
Ensemble predictions
learning_rate=0.1, n_estimators=200
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.4
−0.2
0.0
0.2
0.4

Acknowledgement

“StatQuest with Josh Starmer.” Since 2016, Josh has used an innovative and unique visual style to clearly explain Statistics, Data Science and Machine Learning concepts and algorithms to curious people worldwide.