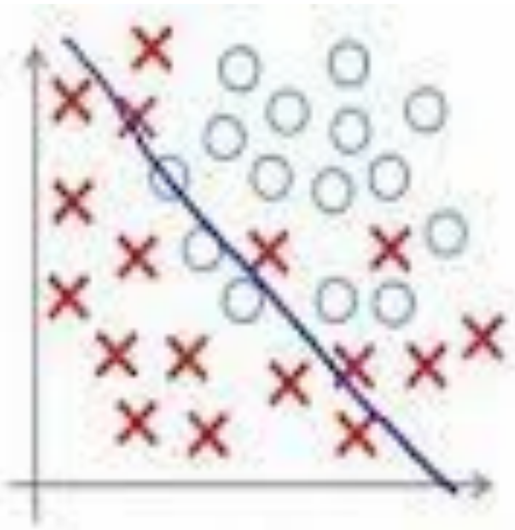
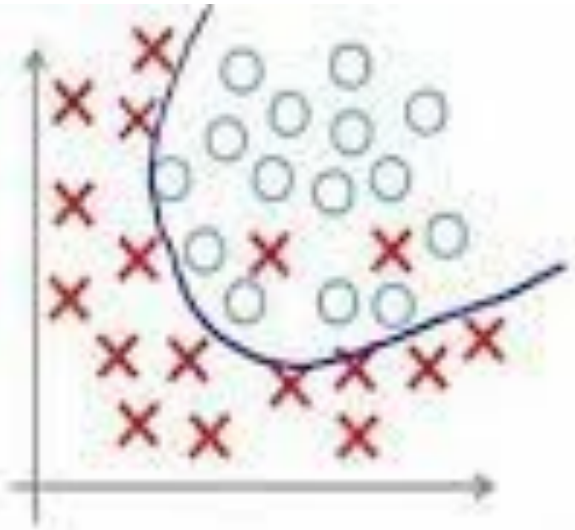
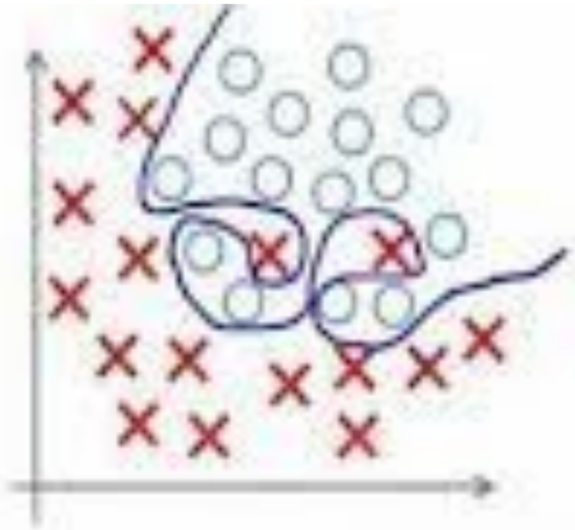
Bias
vs
Variance

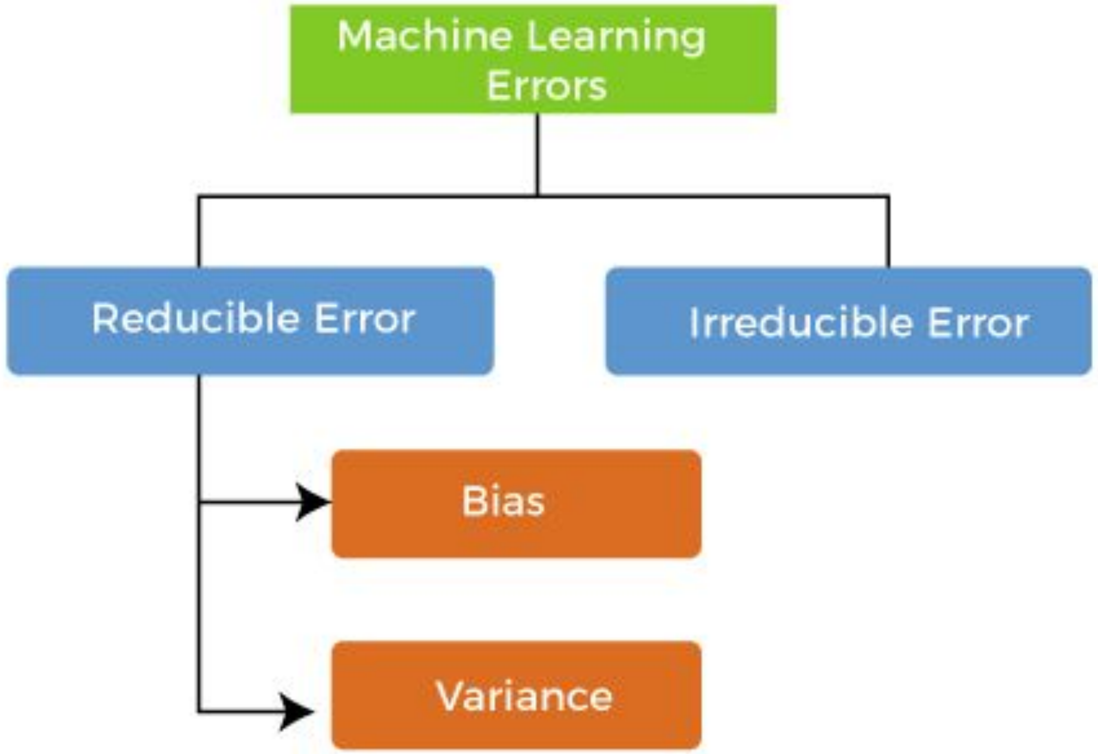

Under-fitting

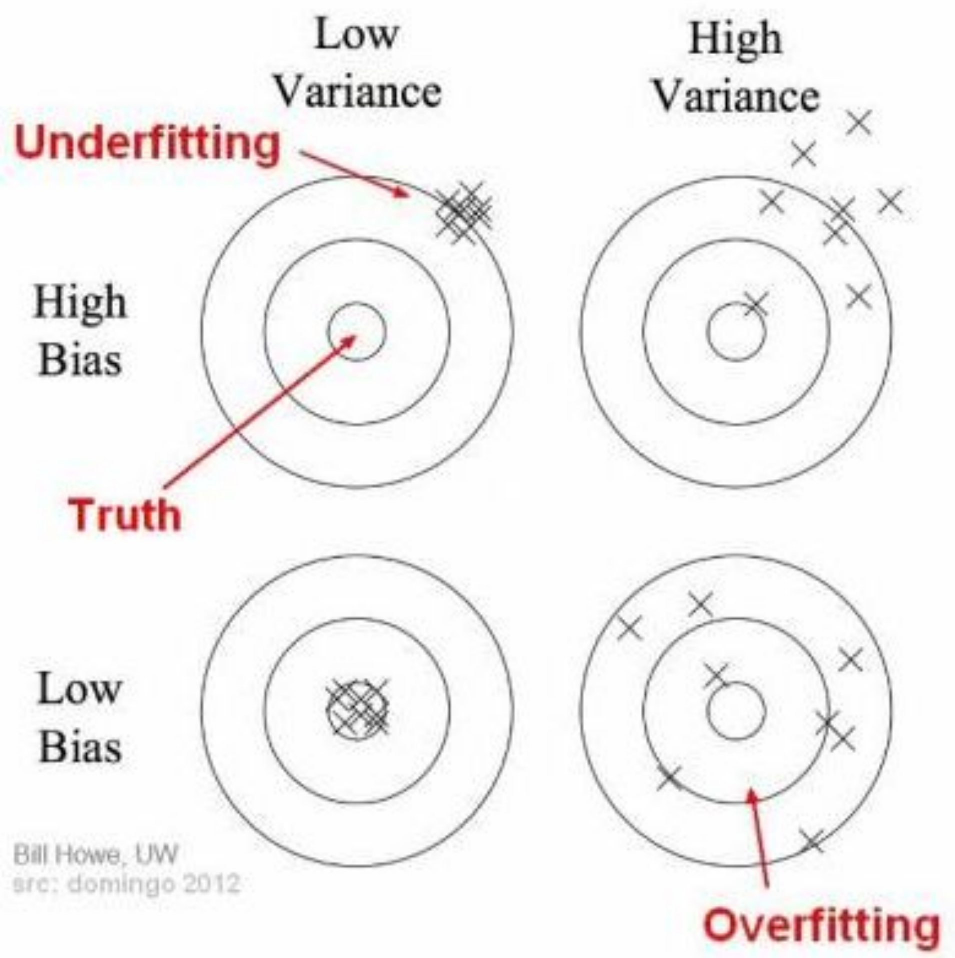
(too simple to explain the variance)


Appropriate-fitting


Over-fitting

(forcefitting – too good to be true)


Machine Learning Errors
Reducible Error
Irreducible Error
Bias
Variance


Low Variance
High Variance
Underfitting
High Bias
Truth
Low Bias
Bill Howe, UW
src: domingo 2012
Overfitting

Ways to reduce High Bias:

High bias mainly occurs due to a much simple model. Below are some ways to reduce the high bias:

- Increase the input features as the model is underfitted.
- Use more complex models, such as including some polynomial features.

A model that exhibits small variance and high bias will underfit the target, while a model with high variance and little bias will overfit the target.

- **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

Bias

Let's assume we have trained the model and are trying to predict values with input 'x_train'. The predicted values are y_predicted. Bias is the error rate of y_predicted and y_train.

> In simple terms,think of bias as the error rate of the training data.

When the error rate is high, we call it High Bias and when the error rate is low, we call it Low Bias

Variance

Let's assume we have trained the model and this time we are trying to predict values with input 'x_test'. Again, the predicted values are y_predicted. Variance is the error rate of the y_predicted and y_test

> In simple terms, think of variance as the error rate of the testing data.

When the error rate is high, we call it High Variance and when the error rate is low, we call it Low Variance

Underfitting

When the model has a high error rate in the training data, we can say the model is underfitting. This usually occurs when the number of training samples is too low. Since our model performs badly on the training data, it consequently performs badly on the testing data as well.

A high error rate in training data implies a High Bias, therefore

In simple terms, High Bias implies underfitting

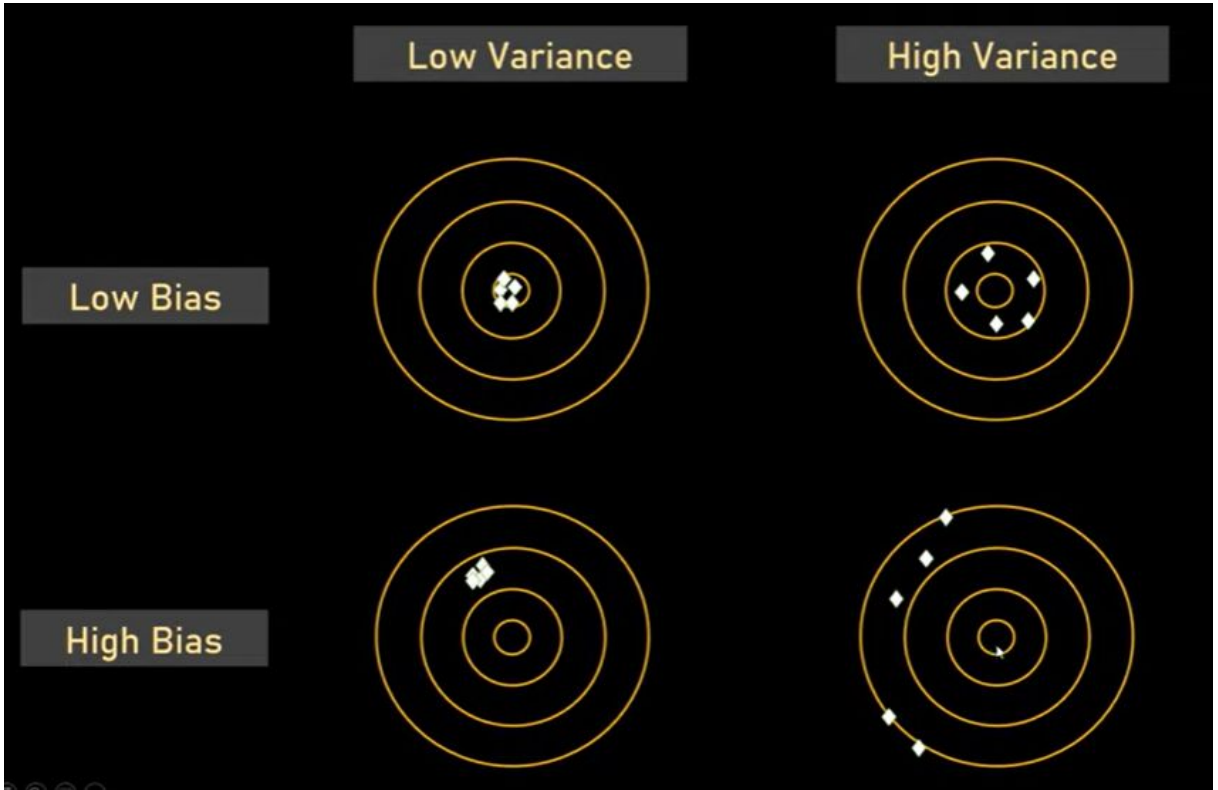
OverFitting

When the model has a low error rate in training data but a high error rate in testing data, we can say the model is overfitting. This usually occurs when the number of training samples is too high or the hyperparameters have been tuned to produce a low error rate on the training data.

A low error rate in training data implies Low Bias whereas a high error rate in testing data implies a High Variance, therefore

> In simple terms, Low Bias and Hight Variance implies overfittting

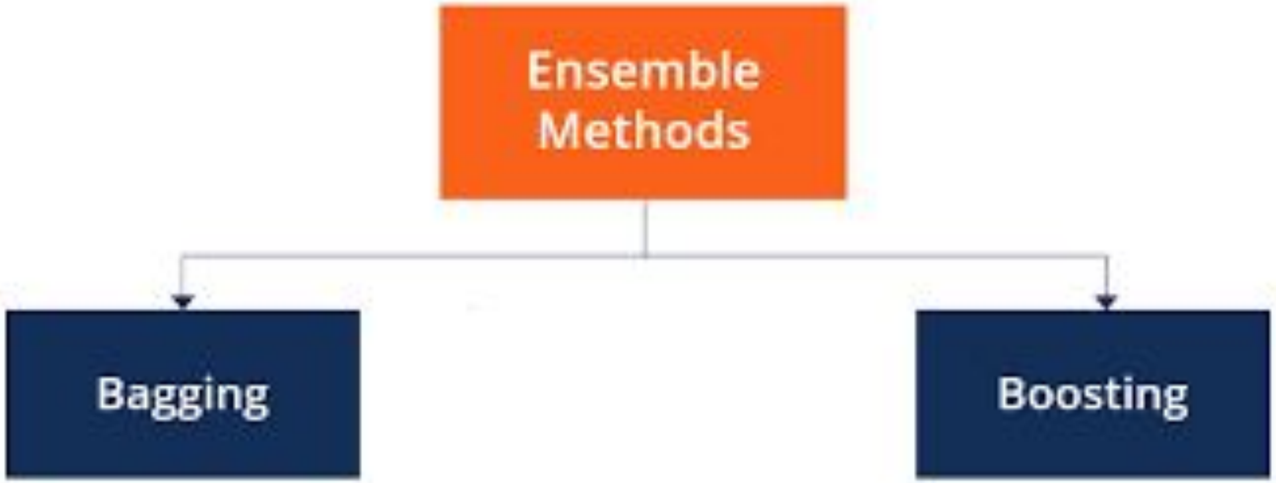
Bulls-eye diagram

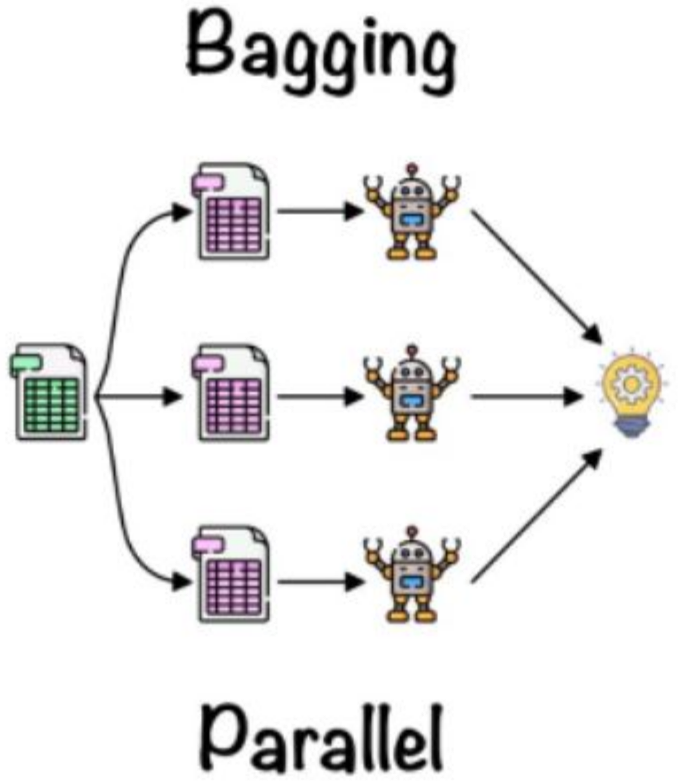


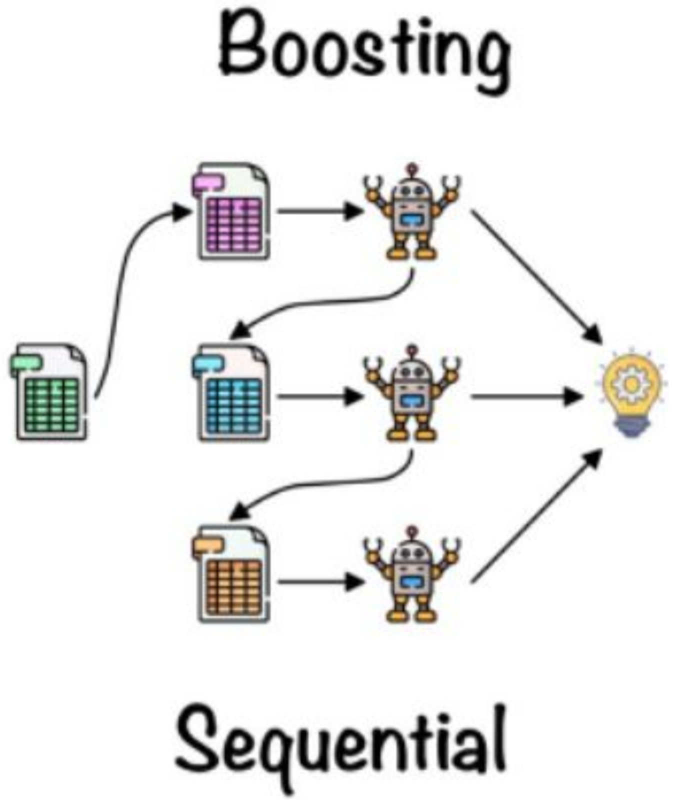
Bagging

Boosting

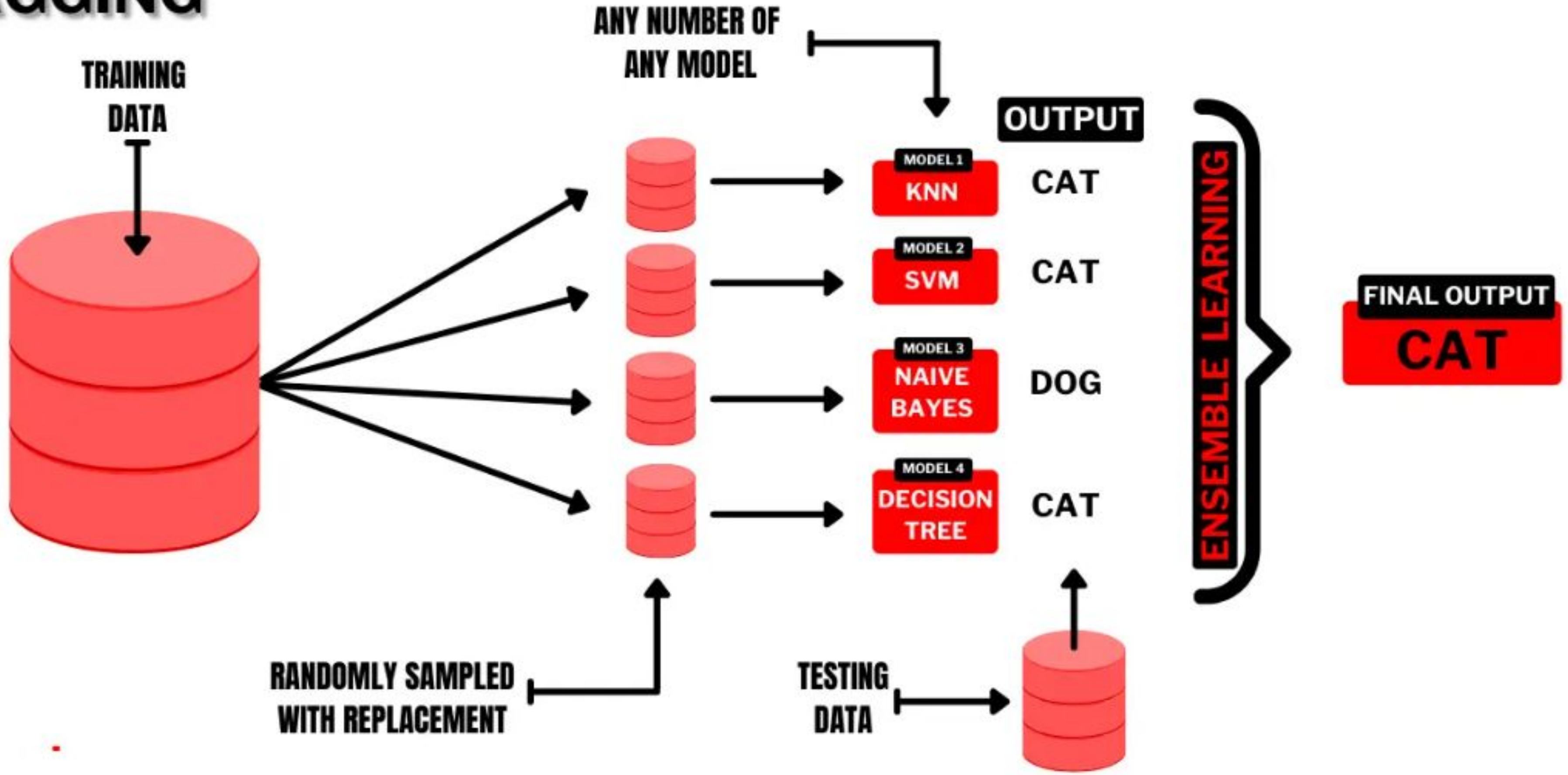
Random Forest


Ensemble Methods
Bagging
Boosting

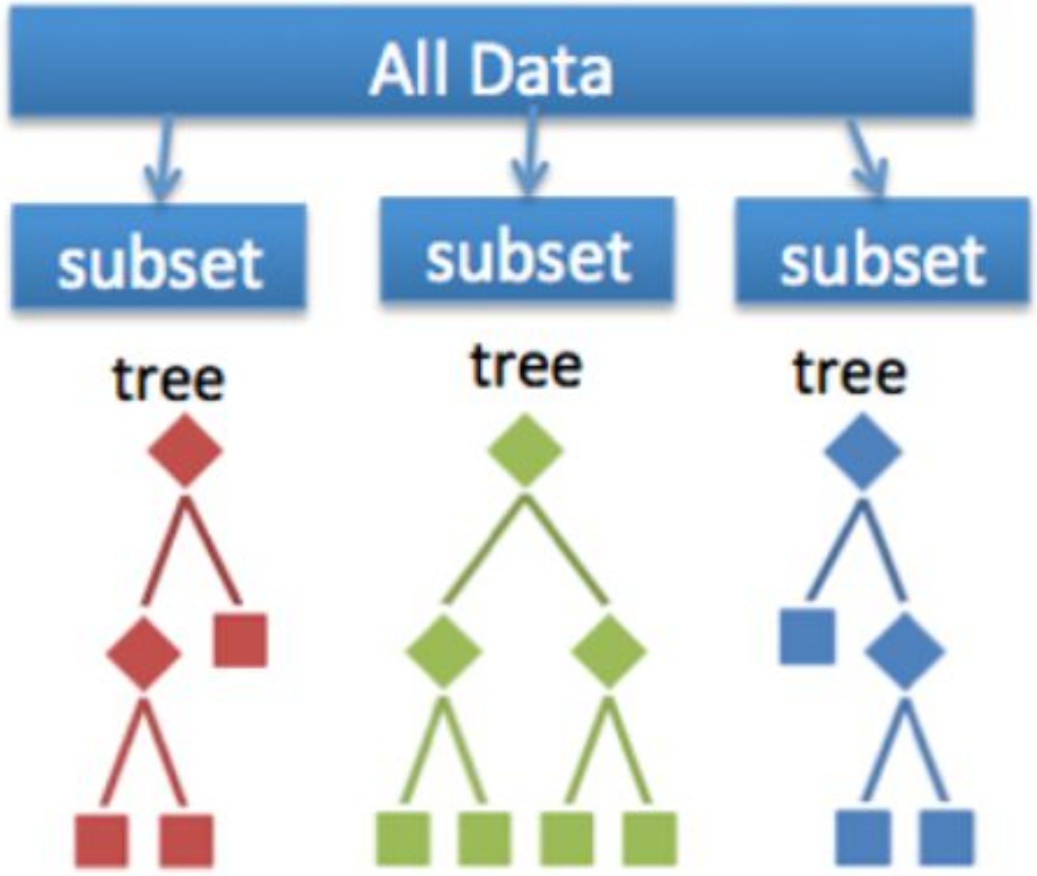

Bagging
Parallel


Boosting
Sequential

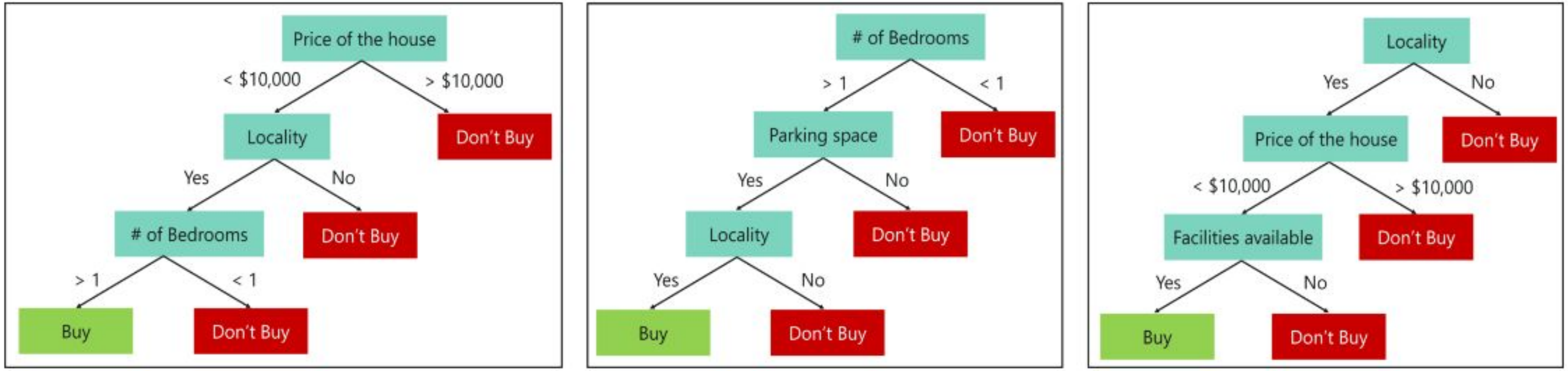
BAGGING


ANY NUMBER OF ANY MODEL
TRAINING DATA
OUTPUT
MODEL 1
KNN
CAT
MODEL 2
SVM
CAT
MODEL 3
NAIVE BAYES
DOG
MODEL 4
DECISION TREE
CAT
ENSEMBLE LEARNING
FINAL OUTPUT
CAT
RANDOMLY SAMPLED WITH REPLACEMENT
TESTING DATA

Random Forest - A Bagging technique

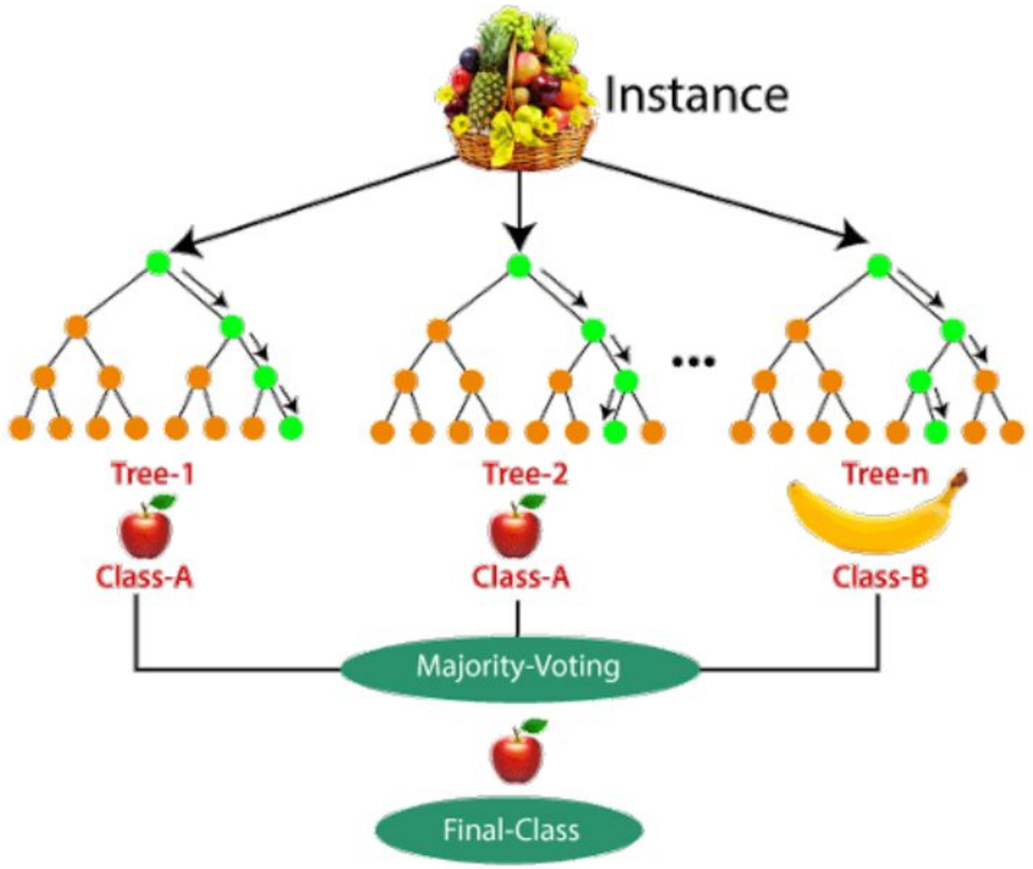


Random Forest - A Bagging technique



Random Forest With 3 Decision Trees

Random Forest - A Bagging technique



Difference Between Decision Tree and Random Forest

Random forest is a collection of decision trees; still, there are a lot of differences in their behavior.

Decision trees	**Random Forest**
1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control.	1. Random forests are created from subsets of data, and the final output is based on average or majority ranking; hence the problem of overfitting is taken care of.
2. A single decision tree is faster in computation.	2. It is comparatively slower.

1. **Bagging**– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.

2. **Boosting**– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST.

Hyperparameters to Increase the Predictive Power of RF

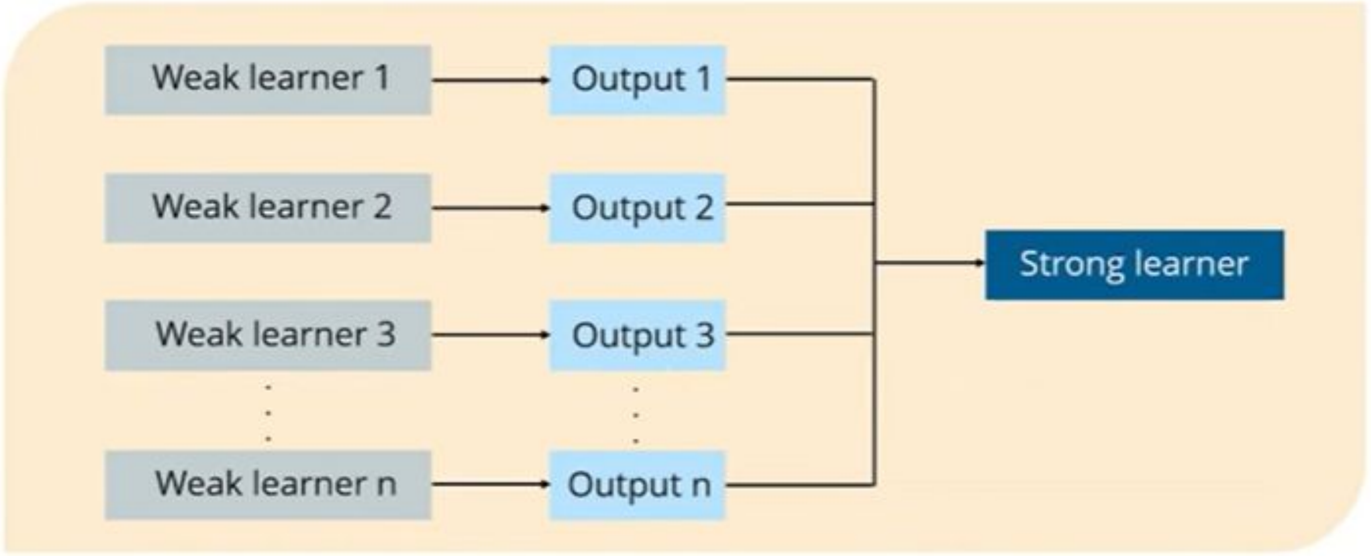
- n_estimators: Number of trees the algorithm builds before averaging the predictions.
- max_features: Maximum number of features random forest considers splitting a node.
- mini_sample_leaf: Determines the minimum number of leaves required to split an internal node.
- criterion: How to split the node in each tree? (Entropy/Gini impurity/Log Loss)
- max_leaf_nodes: Maximum leaf nodes in each tree

Bagging

- Bagging, also known as *Bootstrap Aggregation,* is the ensemble technique used by random forest.
- Each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as *row sampling*. This step of row sampling with replacement is called *bootstrap*.
- Each model is trained independently, which generates results.
- The final output is based on majority voting after combining the results of all models.
- This step which involves combining all the results and generating output based on majority voting, is known as *aggregation*.

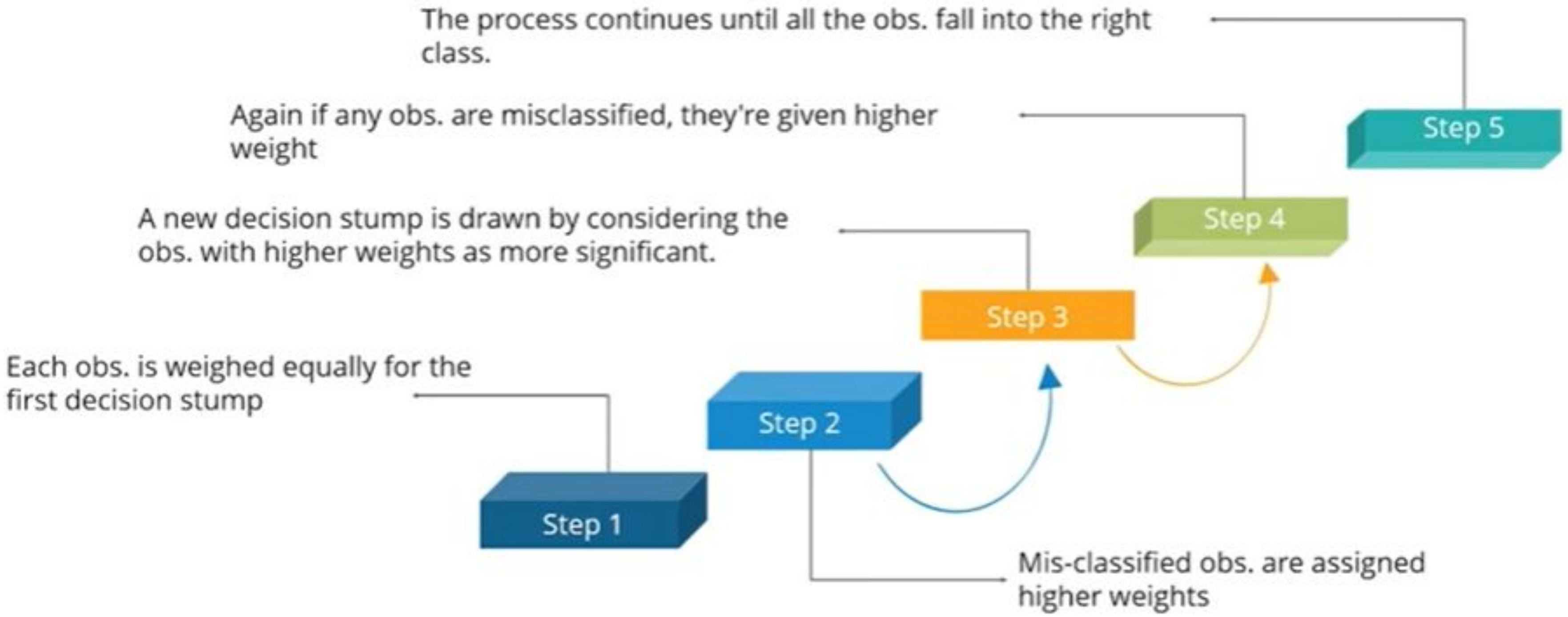
WHAT IS BOOSTING?

Boosting is a process that uses a set of Machine Learning algorithms to combine weak learner to form strong learners in order to increase the accuracy of the model.



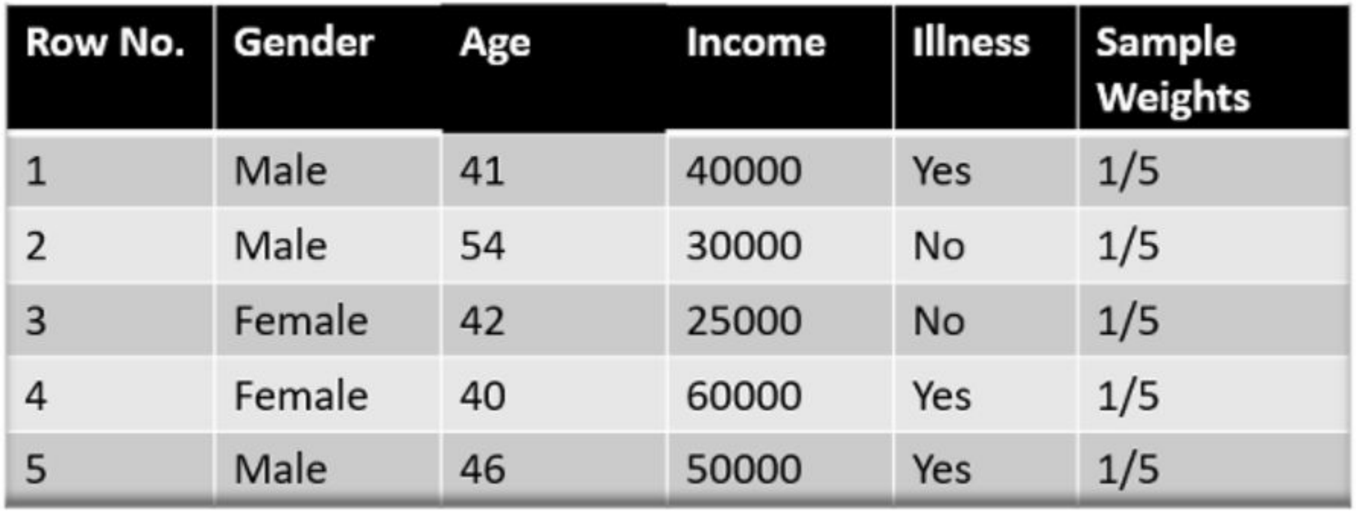
TYPES OF BOOSTING

ADAPTIVE BOOSTING


The process continues until all the obs. fall into the right class.
Again if any obs. are misclassified, they're given higher weight
A new decision stump is drawn by considering the obs. with higher weights as more significant.
Each obs. is weighed equally for the first decision stump
Step 5
Step 4
Step 3
Step 2
Step 1
Mis-classified obs. are assigned higher weights

Adaboost

Step 1 – The Image shown below is the actual representation of our dataset. Since the target column is binary, it is a classification problem. First of all, these data points will be assigned some weights. Initially, all the weights will be equal.



Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

Adaboost

Let’s say Gender has the lowest gini index, so it will be our first stump.

Influence for this classifier in classifying the data points using this formula:

$$Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e\left(\frac{1-Total\ Error}{Total\ Error}\right)$$

Adaboost

The total error is nothing but the summation of all the sample weights of misclassified data points.

Here in our dataset, let's assume there is 1 wrong output, so our total error will be 1/5, and the alpha (performance of the stump) will be:

$$Performance\ of\ the\ stump = \frac{1}{2}\log_e\left(\frac{1 - Total\ Error}{Total\ Error}\right)$$

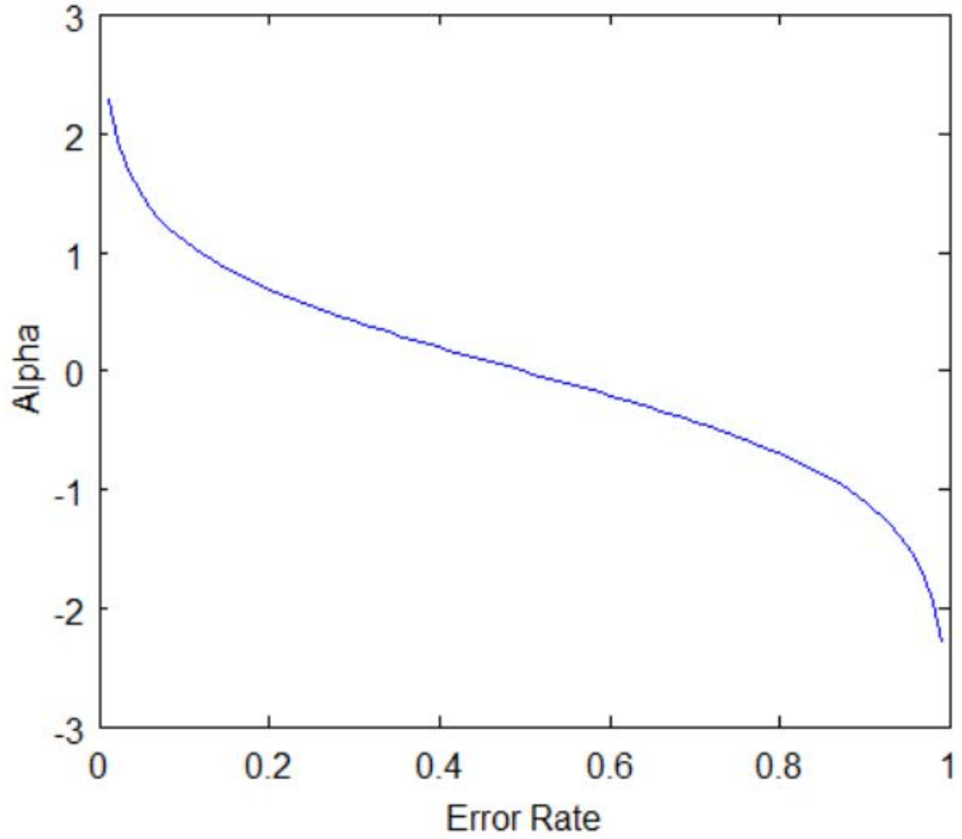
$$\alpha = \frac{1}{2}\log_e\left(\frac{1-\frac{1}{5}}{\frac{1}{5}}\right)$$

$$\alpha = \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

$$\alpha = \frac{1}{2}\log_e(4) = \frac{1}{2} * (1.38) = 0.69$$

Adaboost

0 Indicates perfect stump, and 1 indicates horrible stump.



Adaboost

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

New weights for *correctly classified* samples are:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(-0.69)$$

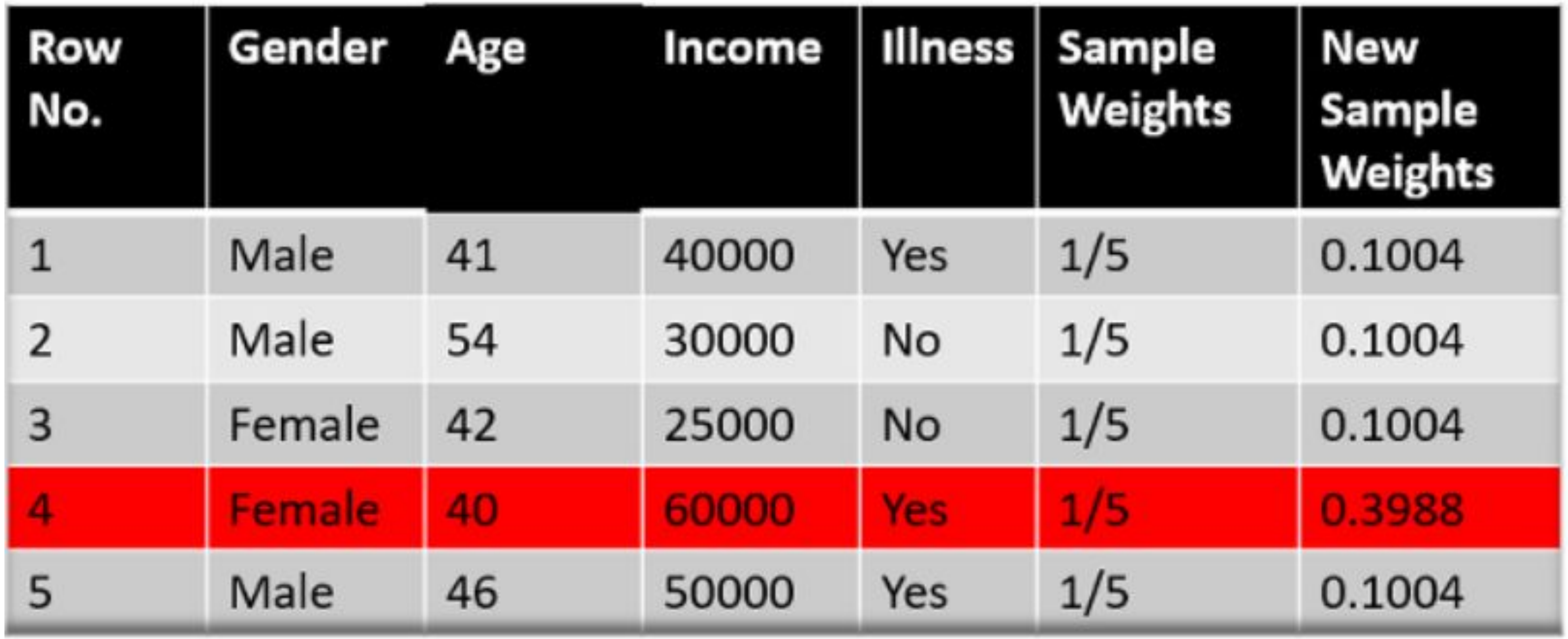
$$New\ sample\ weight\ =\ 0.2\ *\ 0.502\ =\ 0.1004$$

For *wrongly classified* samples, the updated weights will be:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(0.69)$$

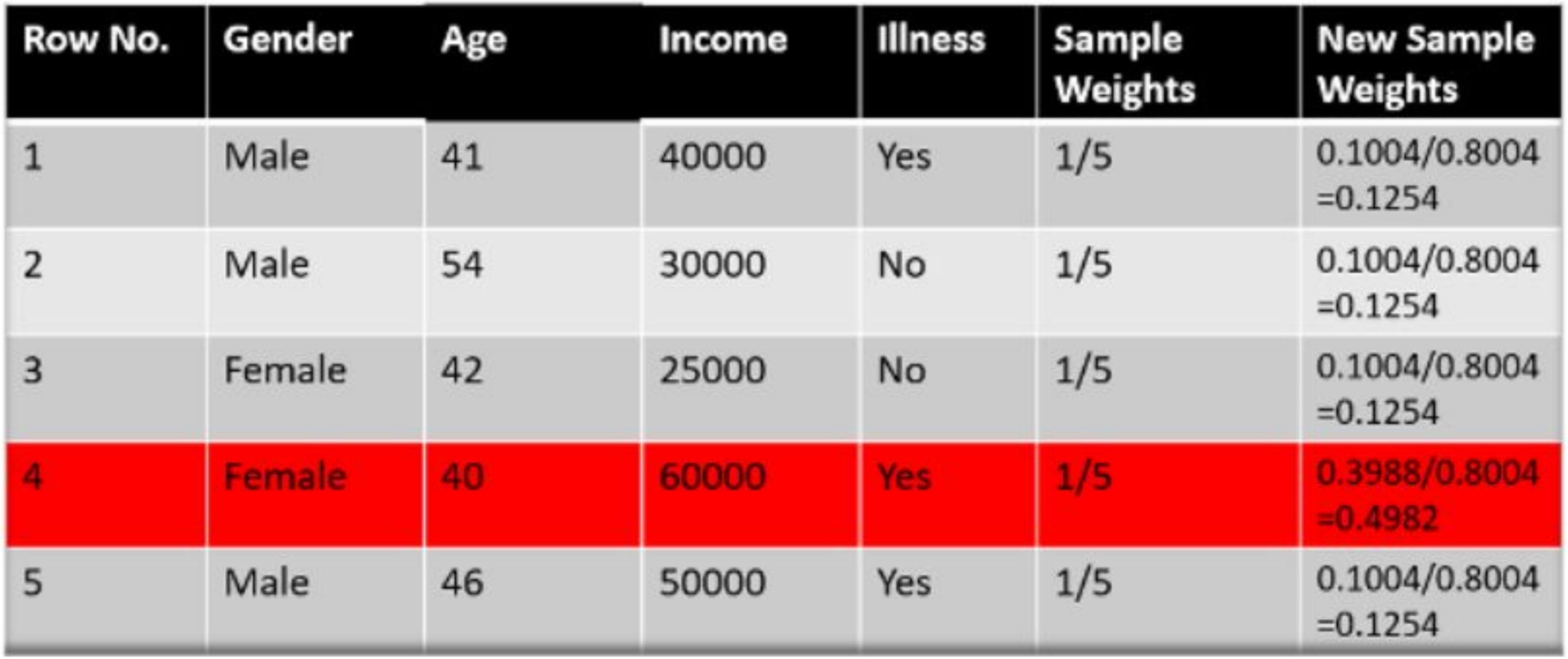
$$New\ sample\ weight\ =\ 0.2\ *\ 1.994\ =\ 0.3988$$

Adaboost



Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

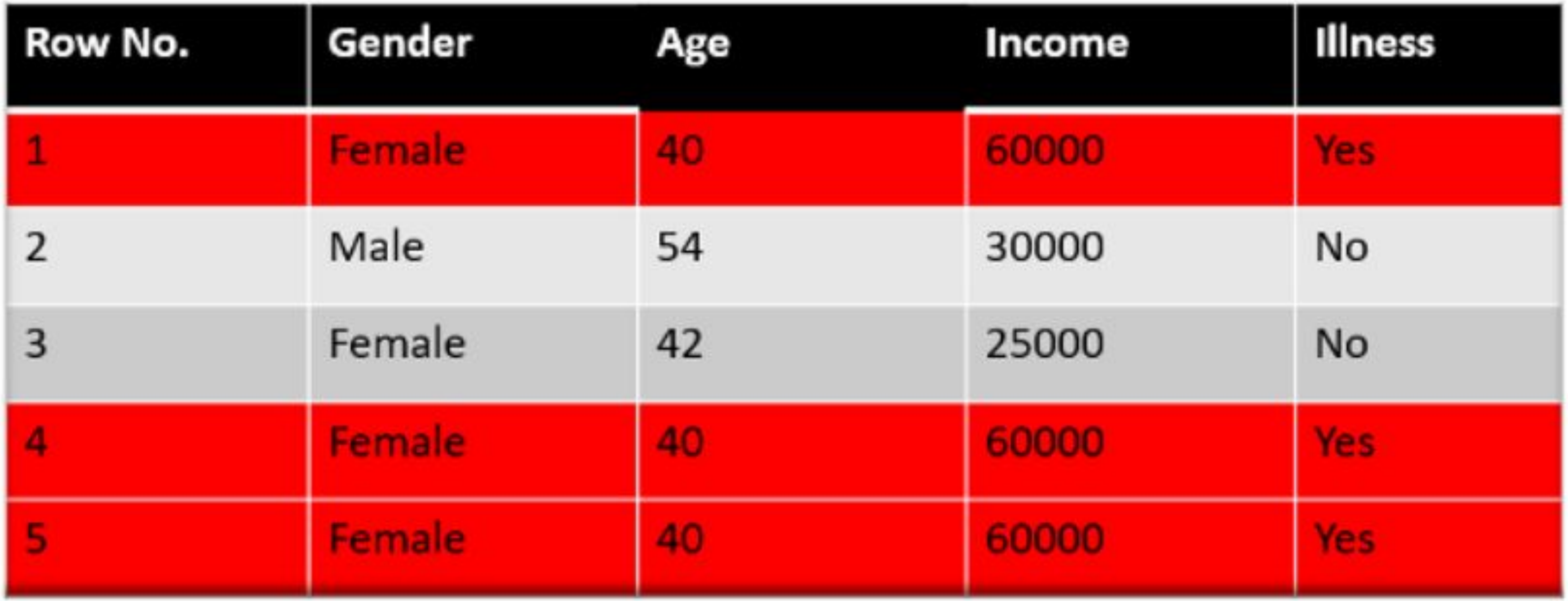
Adaboost



Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

Adaboost

This comes out to be our new dataset, and we see the data point, which was wrongly classified, has been selected 3 times because it has a higher weight.



Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

Adaboost

Now this act as our new dataset, and we need to repeat all the above steps i.e.

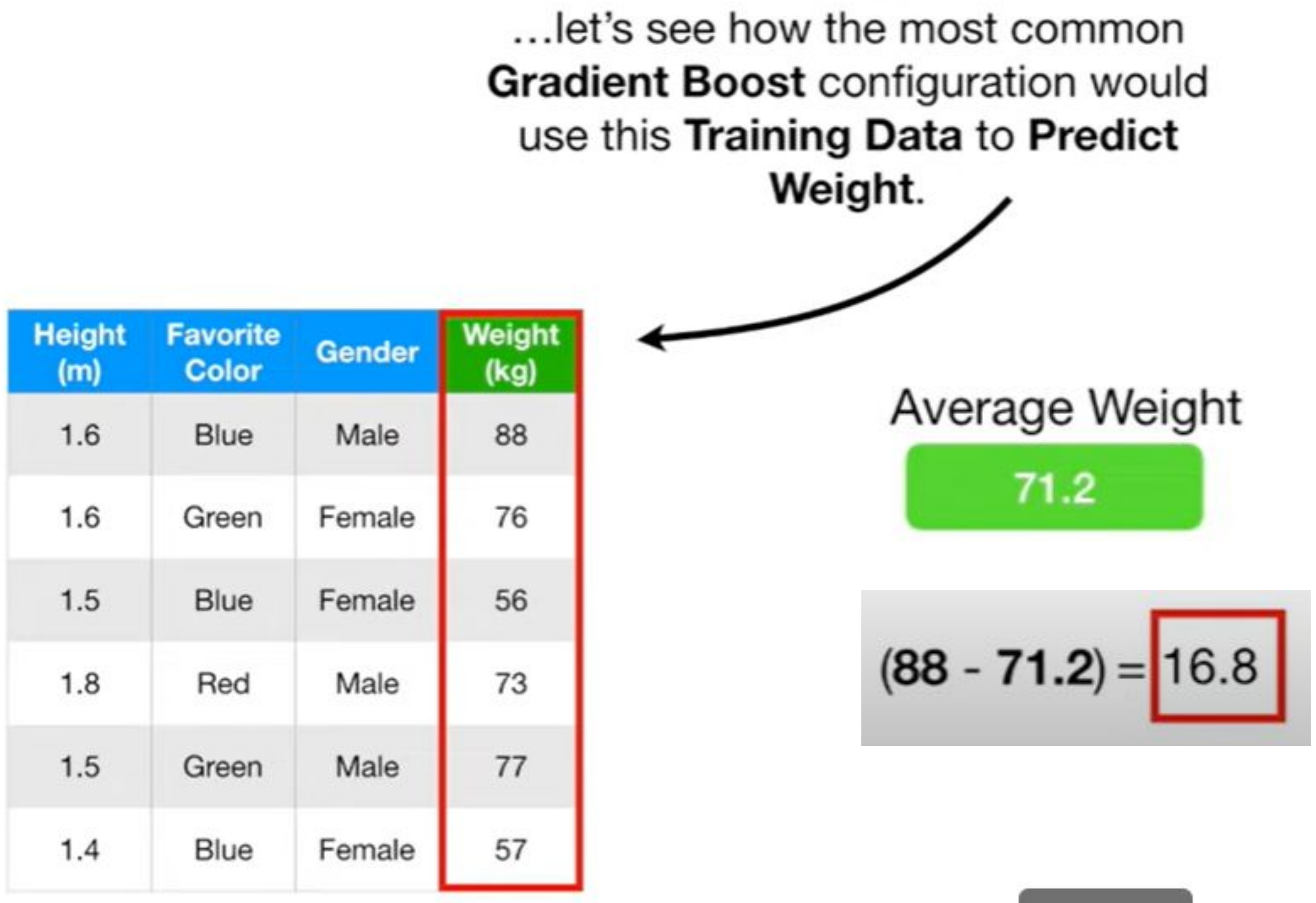
- Assign *equal weights* to all the data points.
- Find the stump that does the *best job classifying* the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index.
- Calculate *“Total error”* to update the previous sample weights.
- Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.

Suppose, with respect to our dataset, we have constructed 3 decision trees (DT1, DT2, DT3) in a *sequential manner.* If we send our test data now, it will pass through all the decision trees, and finally, we will see which class has the majority, and based on that, we will do predictions for our test dataset.

GRADIENT BOOSTING

Gradient Boosting



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

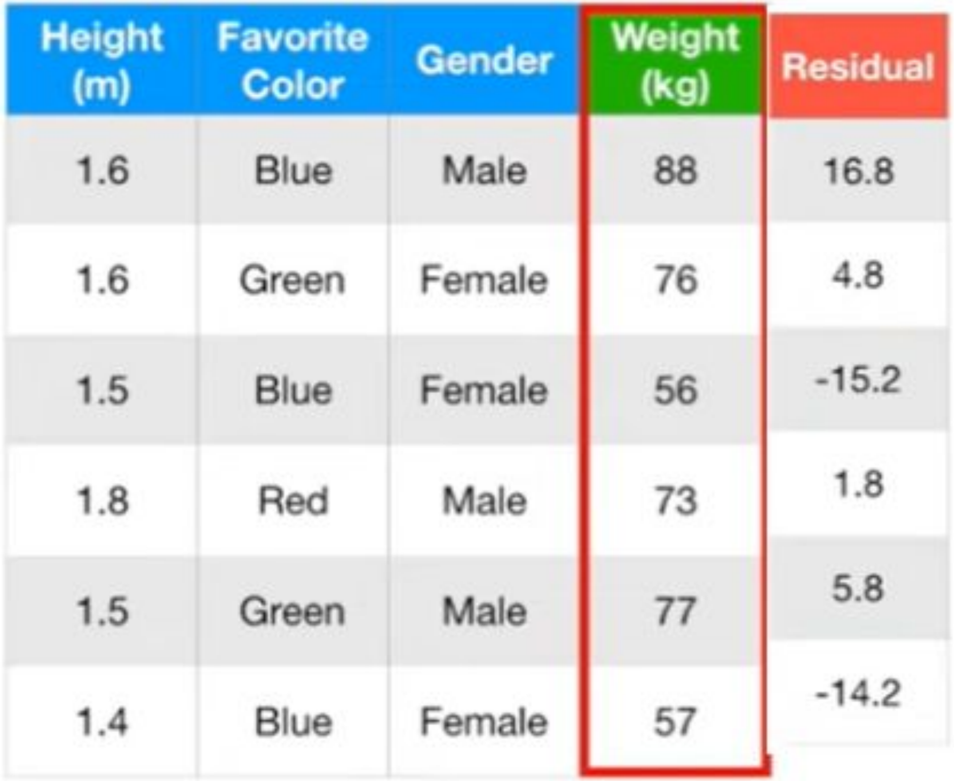
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Average Weight

71.2

$$(88 - 71.2) = \boxed{16.8}$$

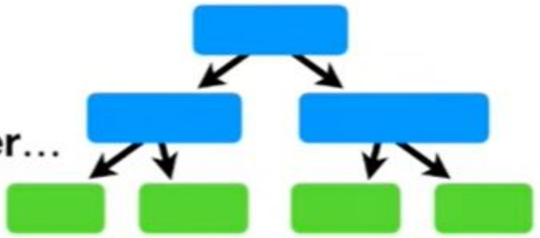
Gradient Boosting



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

Gradient Boosting

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

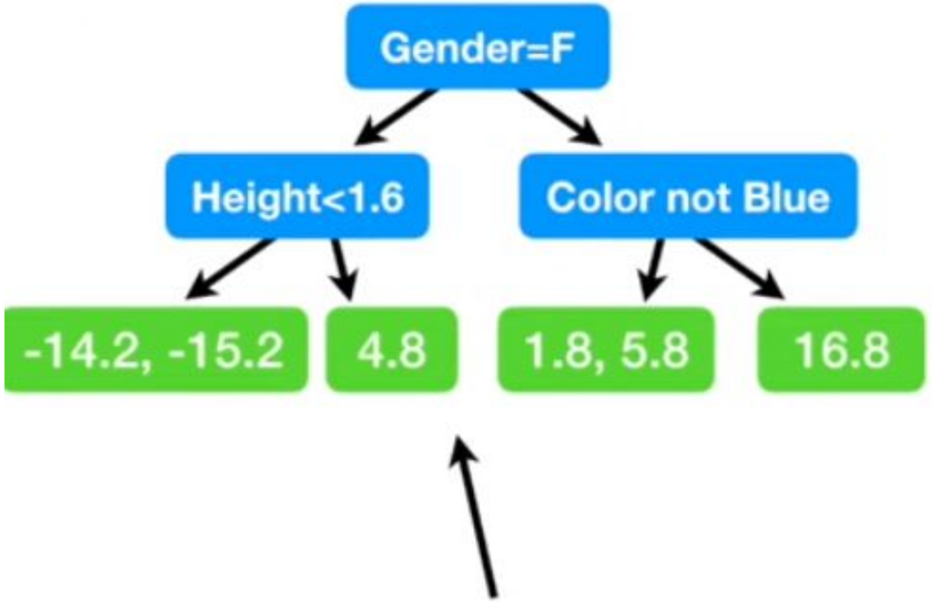


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

...to **Predict** the **Residuals**.

Left Tree = Yes
Right Tree = No

Gradient Boosting



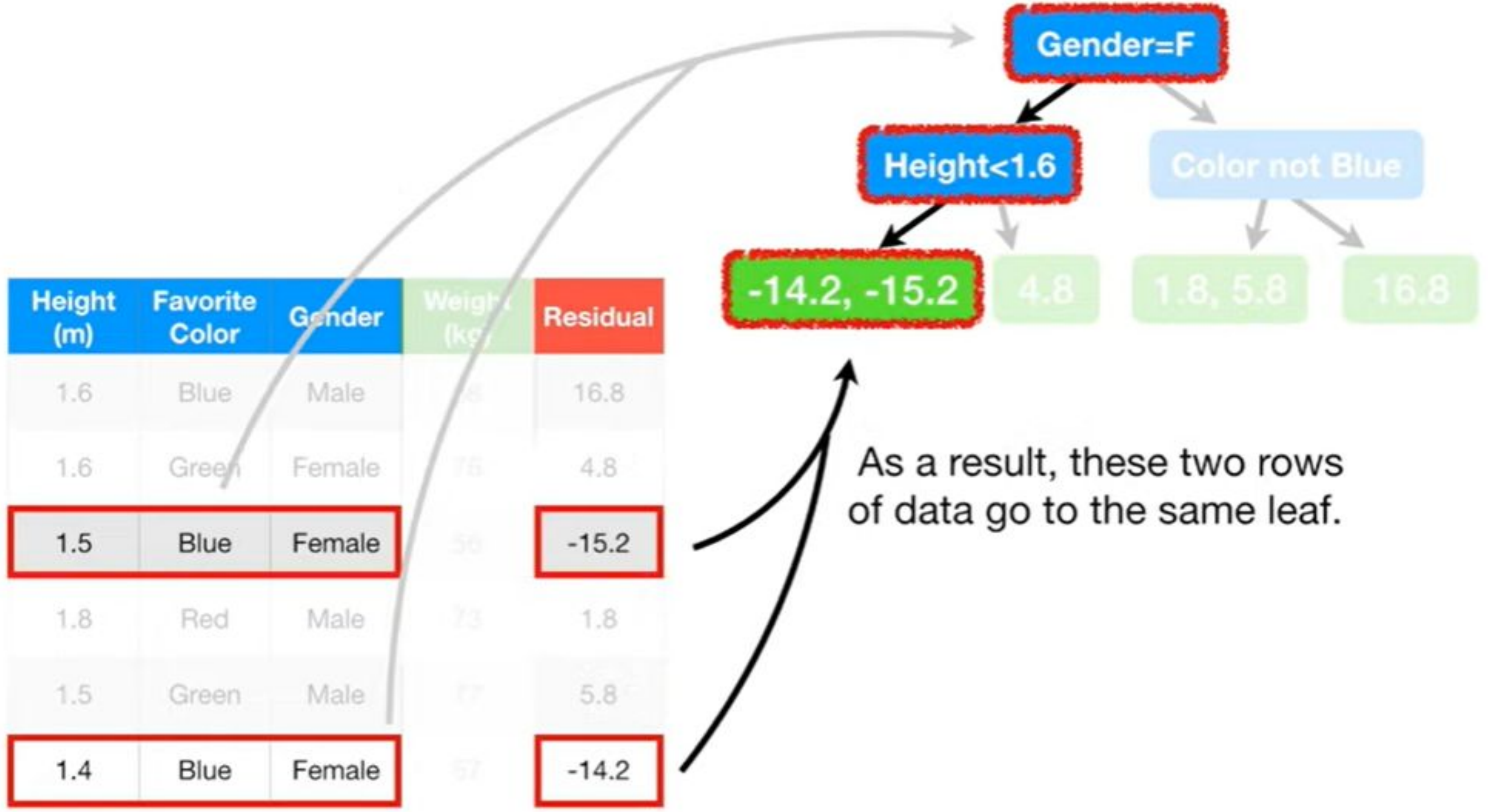
So, setting aside the reason why we are building a tree to **Predict** the **Residuals** for the time being, here's the tree!

Remember, in this example we are only allowing up to four leaves...

...but when using a larger dataset, it is common to allow anywhere from **8** to **32**.

Left Tree = Yes
Right Tree = No

Gradient Boosting

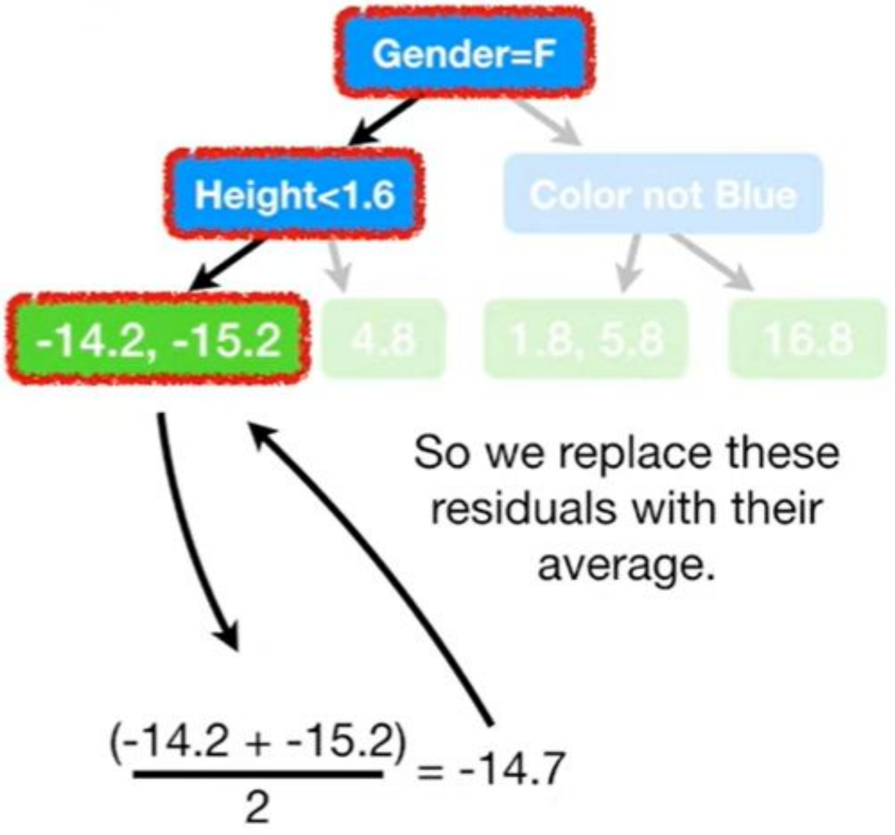


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

As a result, these two rows of data go to the same leaf.

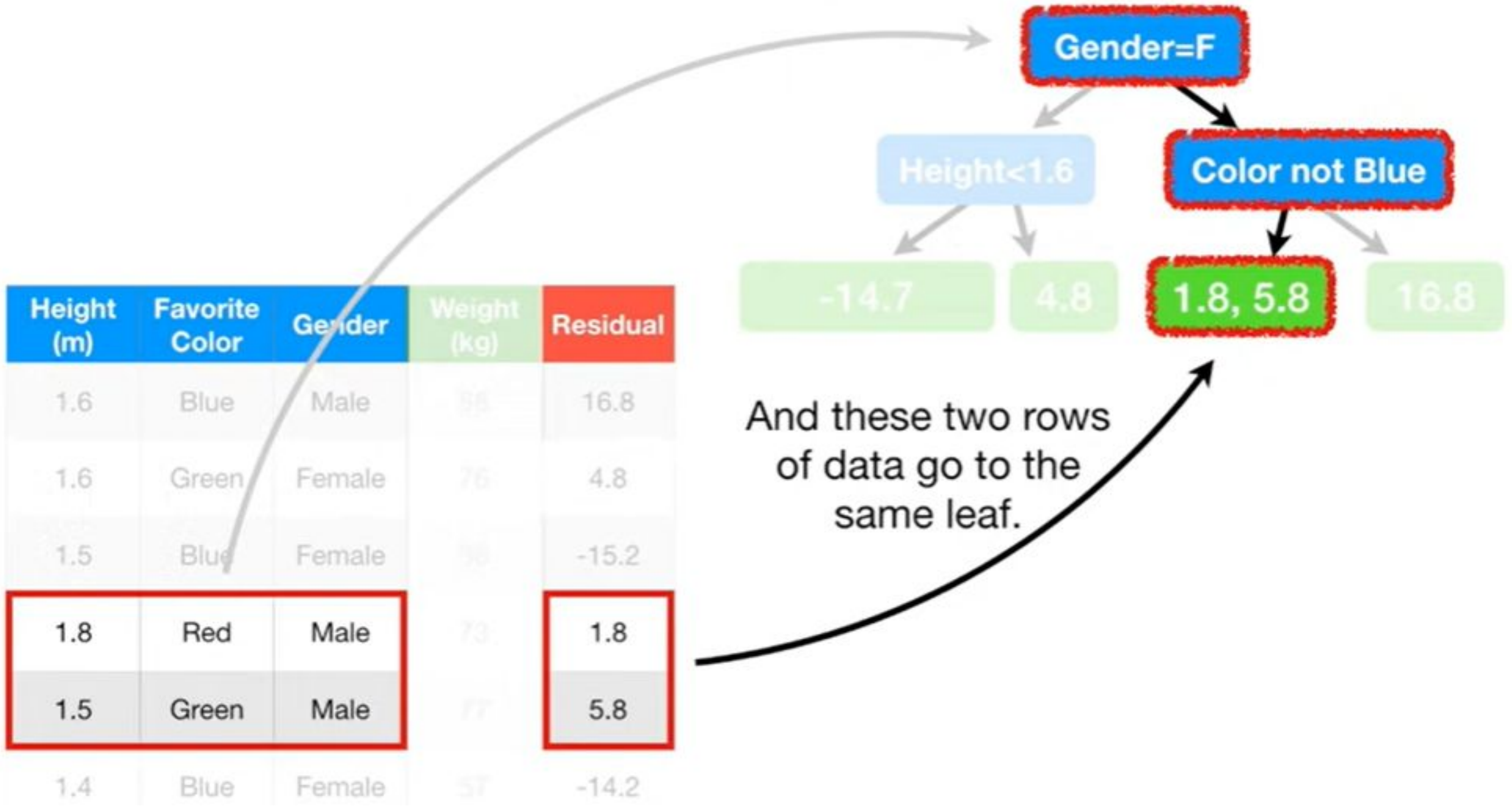
Left Tree = Yes
Right Tree = No

Gradient Boosting


Gender=F
Height<1.6
Color not Blue
-14.2, -15.2
4.8
1.8, 5.8
16.8
So we replace these residuals with their average.
$\frac{(-14.2 + -15.2)}{2} = -14.7$

Left Tree = Yes
Right Tree = No

Gradient Boosting

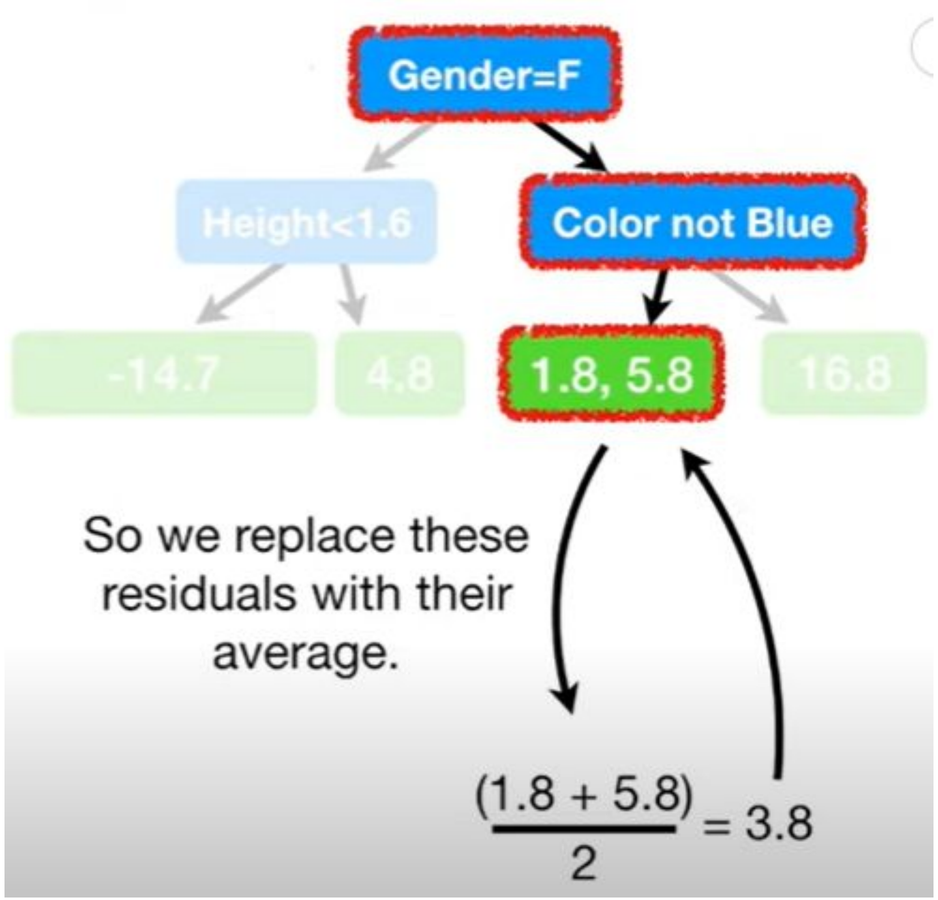


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

And these two rows of data go to the same leaf.

Left Tree = Yes
Right Tree = No

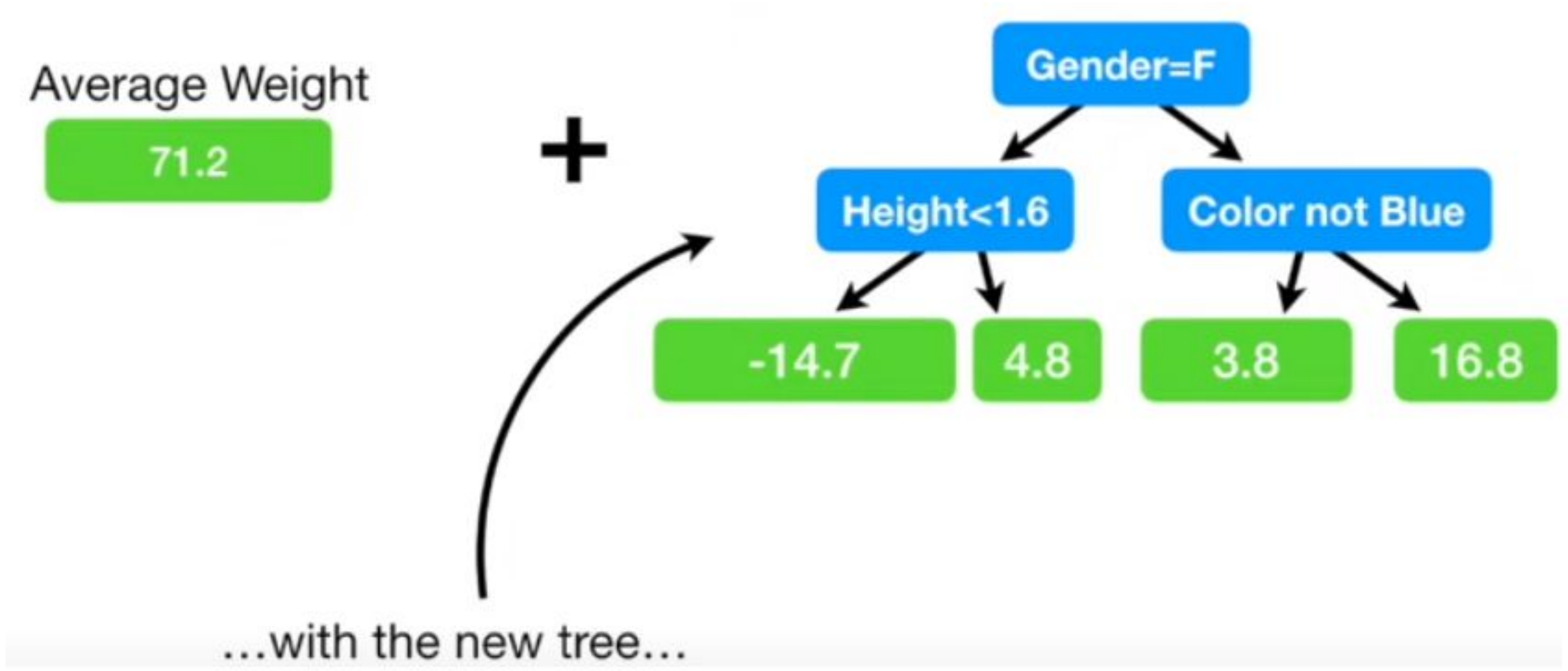
Gradient Boosting



So we replace these residuals with their average.

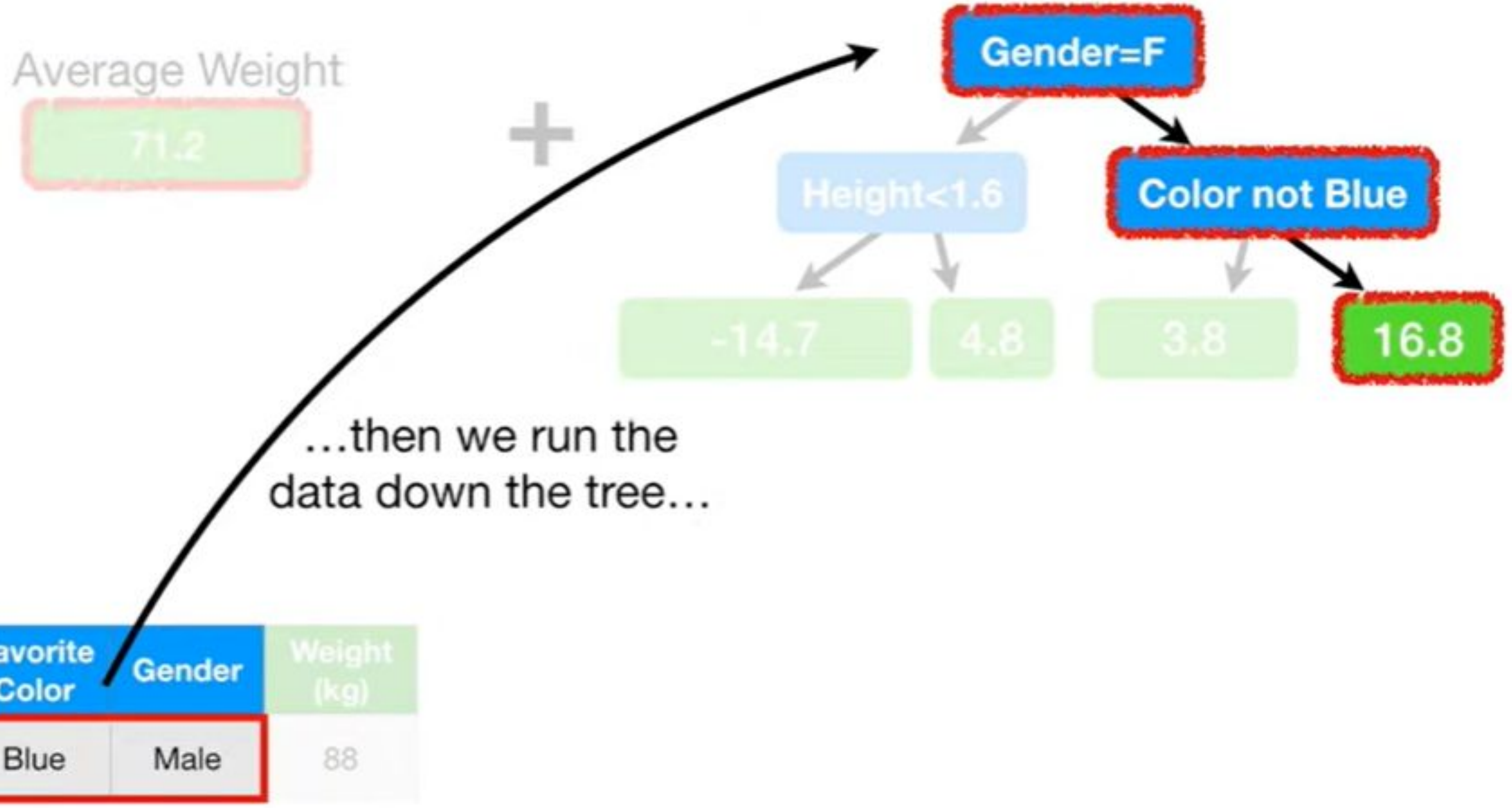
$$\frac{(1.8 + 5.8)}{2} = 3.8$$

Gradient Boosting



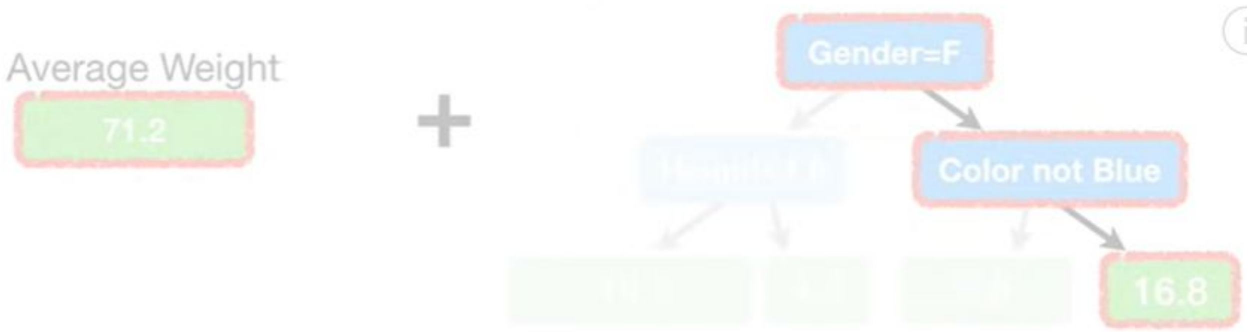
Left Tree = Yes
Right Tree = No

Gradient Boosting



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

Gradient Boosting



Predicted Weight = 71.2 + 16.8 $\boxed{= 88}$

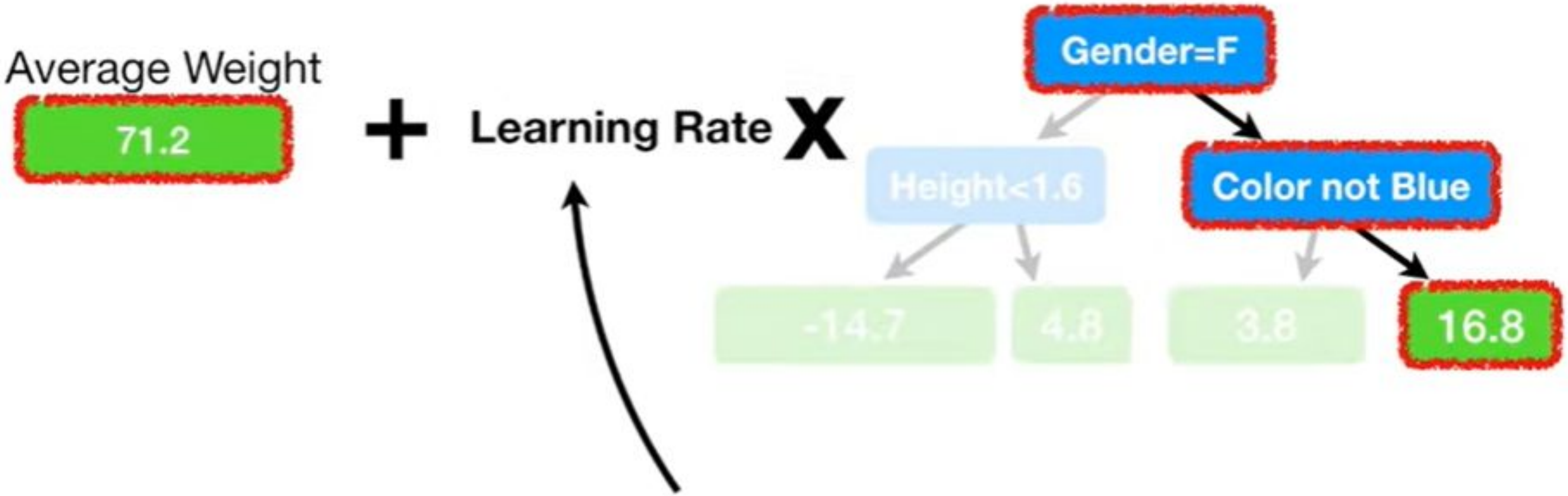
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

…which is the same as the **Observed Weight**.

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

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

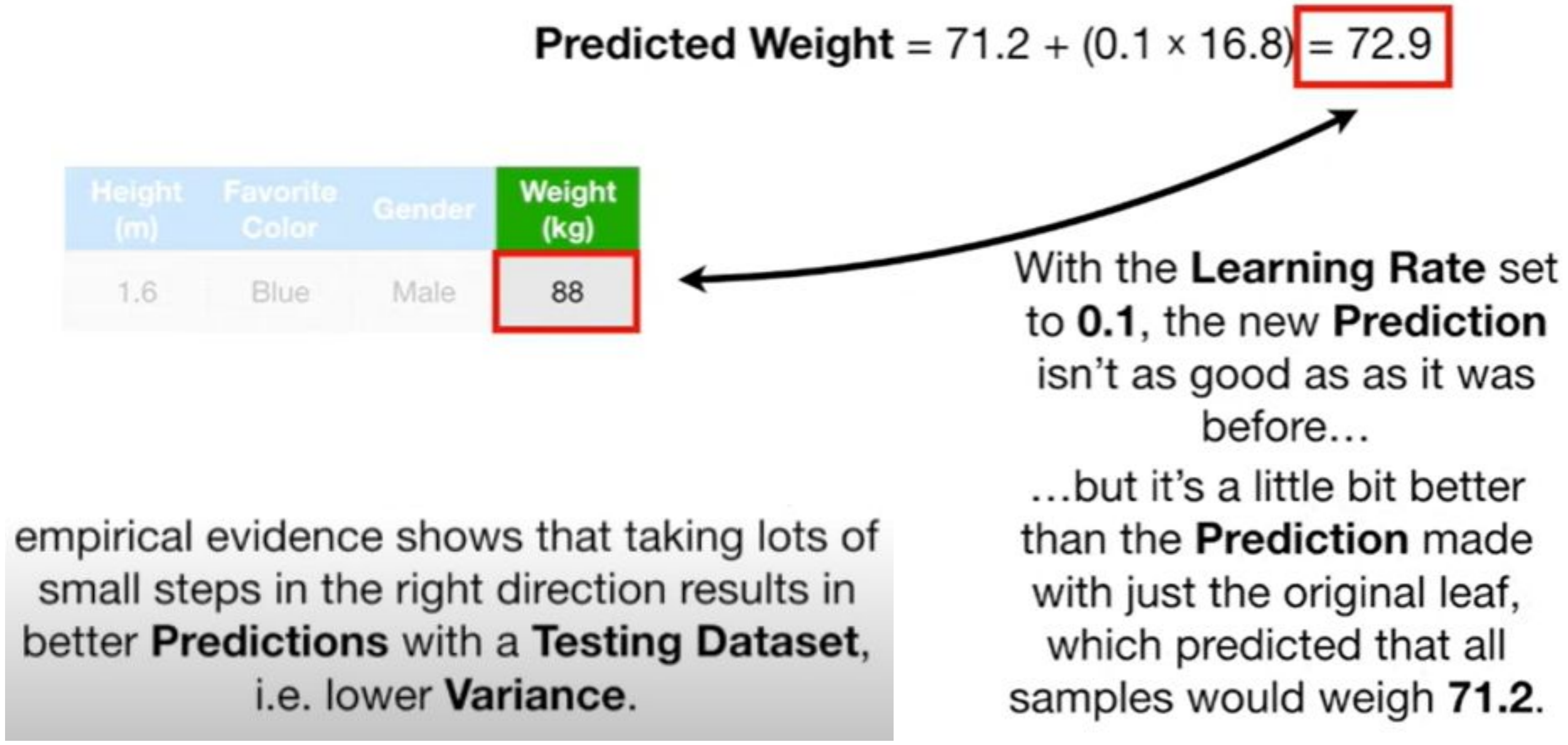
Gradient Boosting



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.

Gradient Boosting



Predicted Weight = 71.2 + (0.1 × 16.8) $\boxed{= 72.9}$

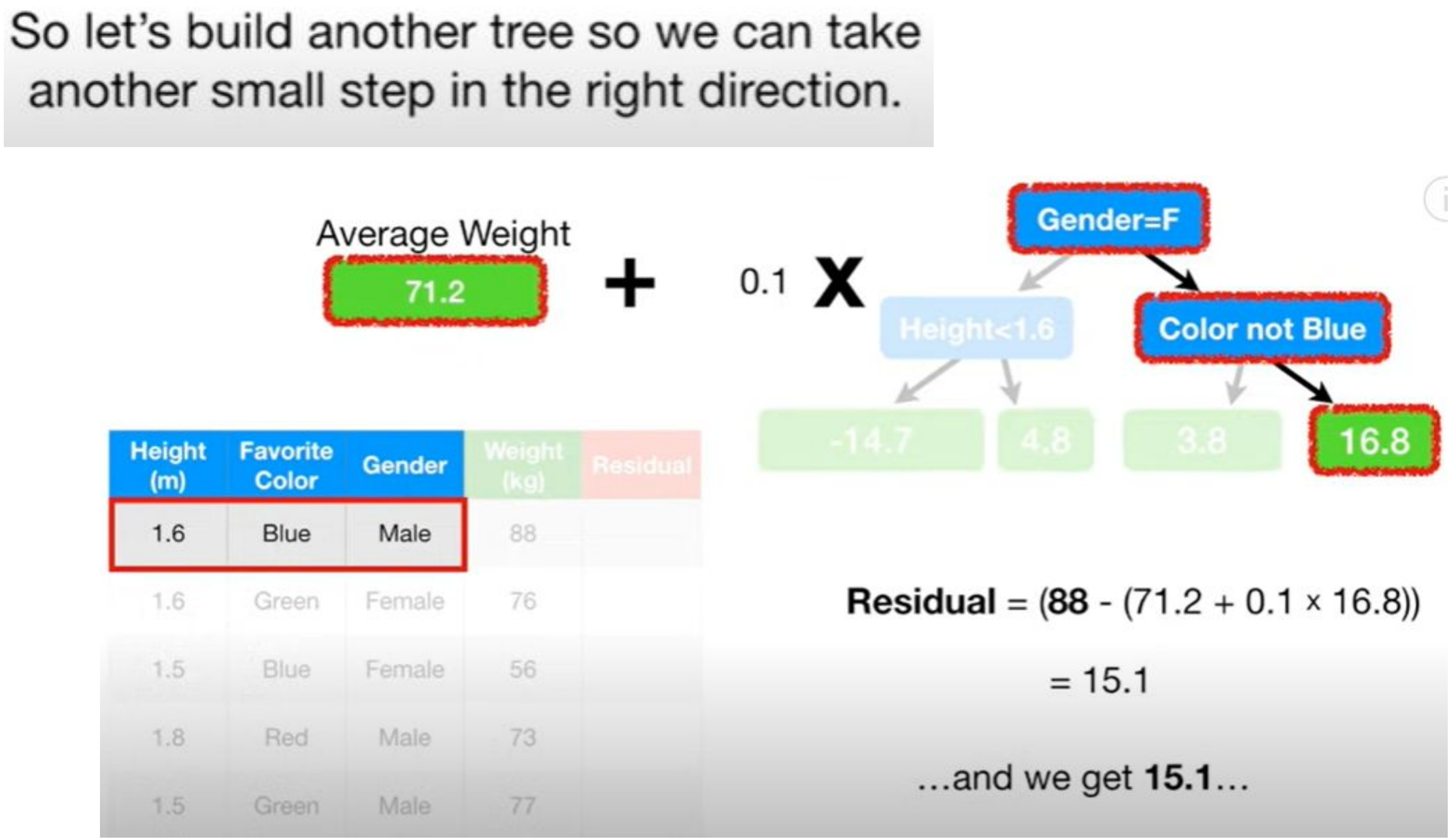
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	$\boxed{88}$

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

…but it's a little bit better than the **Prediction** made with just the original leaf, which predicted that all samples would weigh **71.2**.

empirical evidence shows that taking lots of small steps in the right direction results in better **Predictions** with a **Testing Dataset**, i.e. lower **Variance**.

Gradient Boosting



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

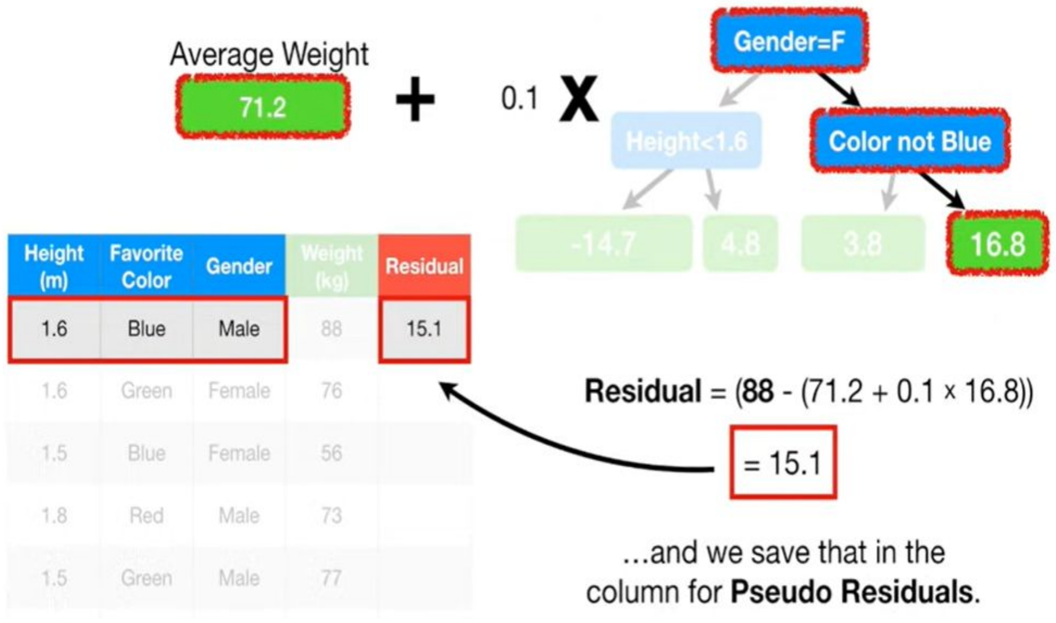
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	

Residual = (**88** - (71.2 + 0.1 × 16.8))

= 15.1

…and we get **15.1**…

Gradient Boosting



Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	

Residual = (**88** - (71.2 + 0.1 × 16.8))

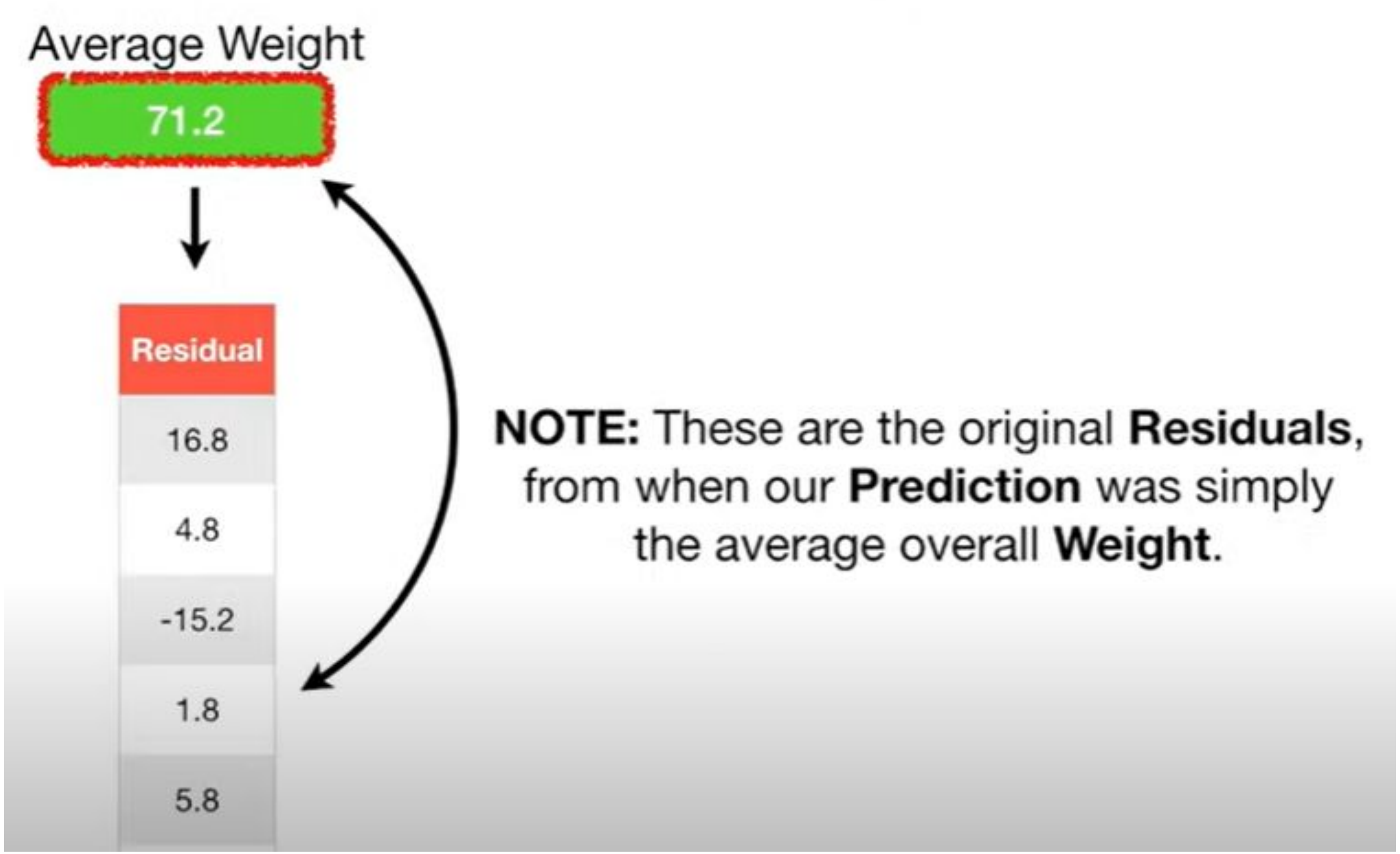
$\boxed{= 15.1}$

...and we save that in the column for **Pseudo Residuals**.

Just like before, we calculate the **Pseudo Residuals**, the difference between the **Observed Weights** and our latest **Predictions**.

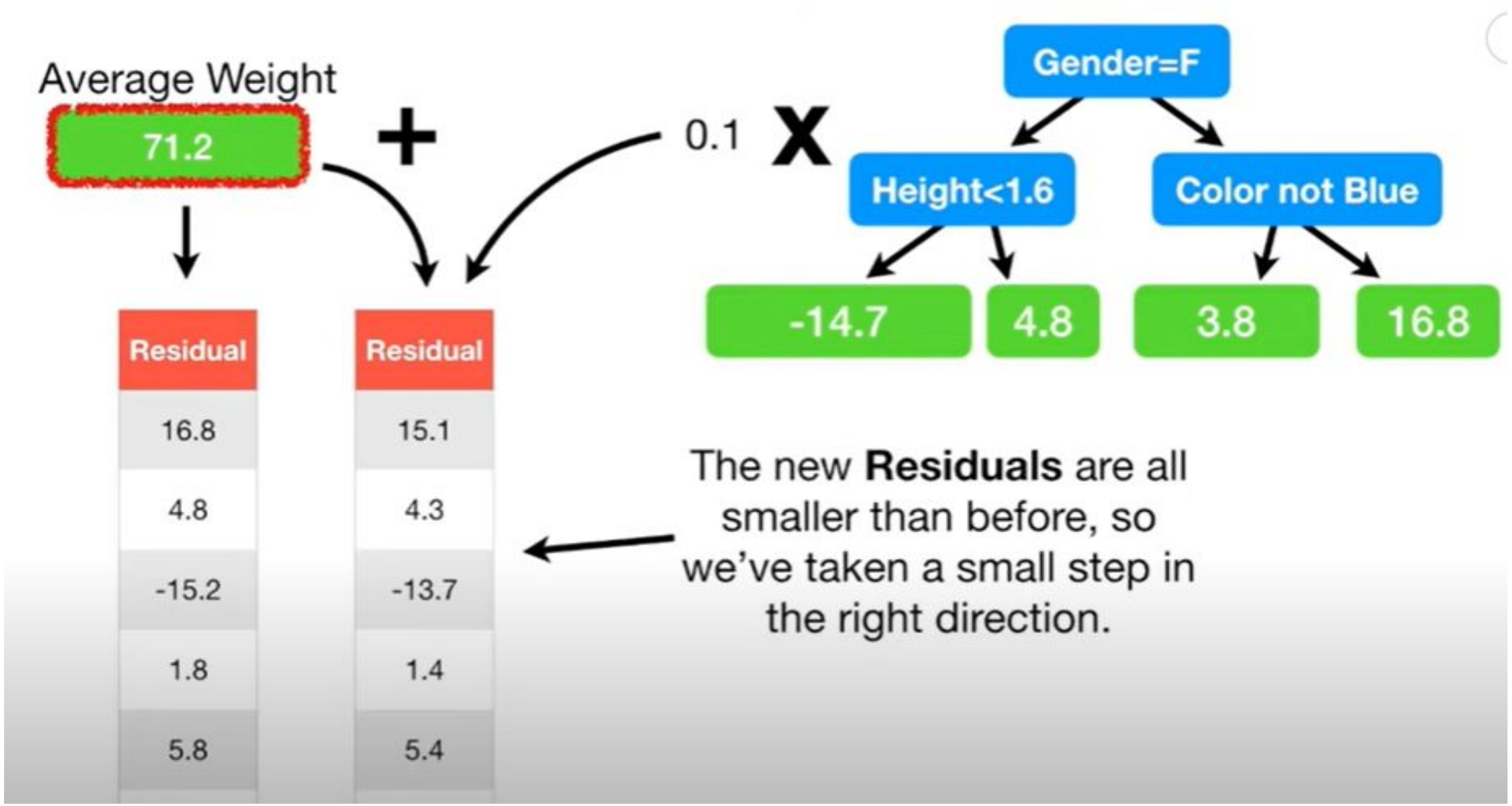
Residual = (**Observed** - **Predicted**)

Gradient Boosting



NOTE: These are the original **Residuals**, from when our **Prediction** was simply the average overall **Weight**.

Gradient Boosting

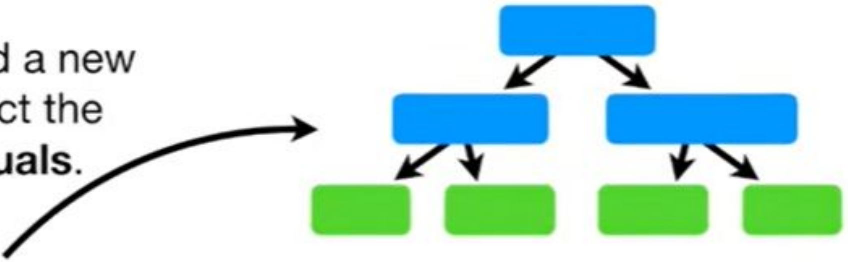


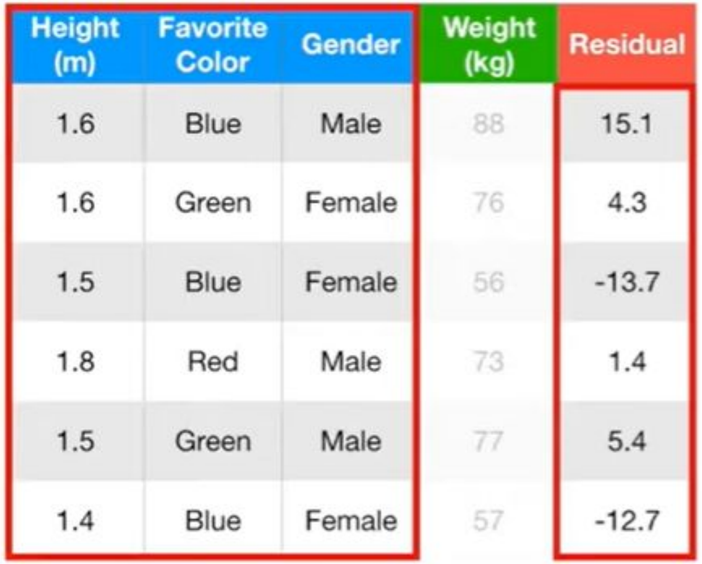
Residual	Residual
16.8	15.1
4.8	4.3
-15.2	-13.7
1.8	1.4
5.8	5.4

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

Gradient Boosting

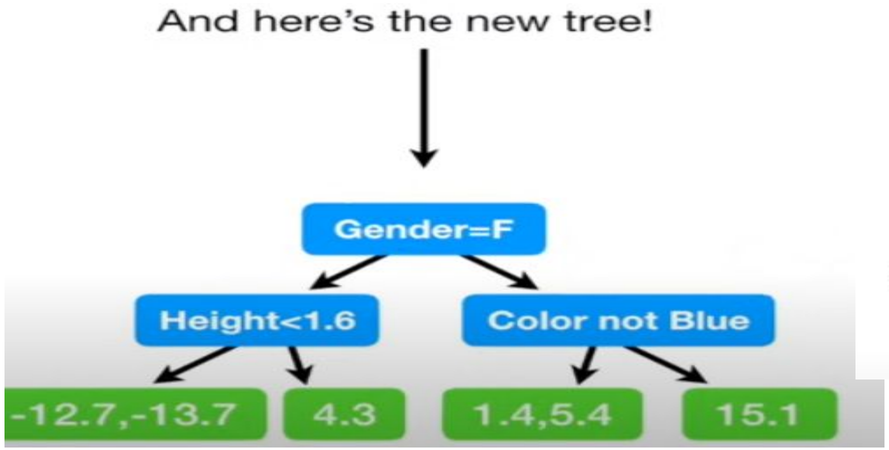
Now let's build a new tree to predict the new **Residuals**.



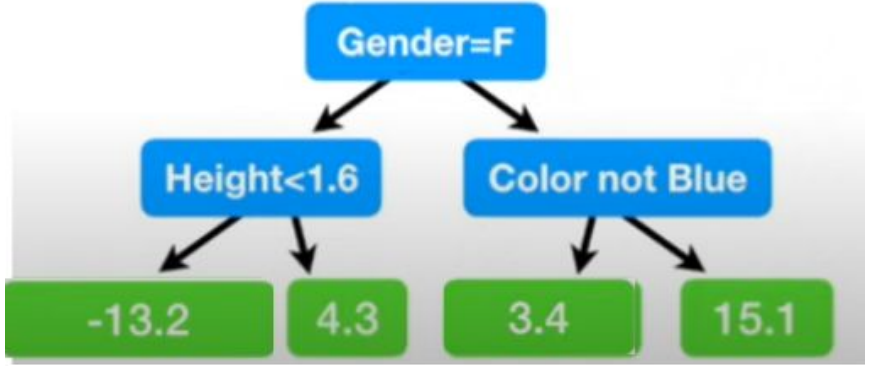


Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

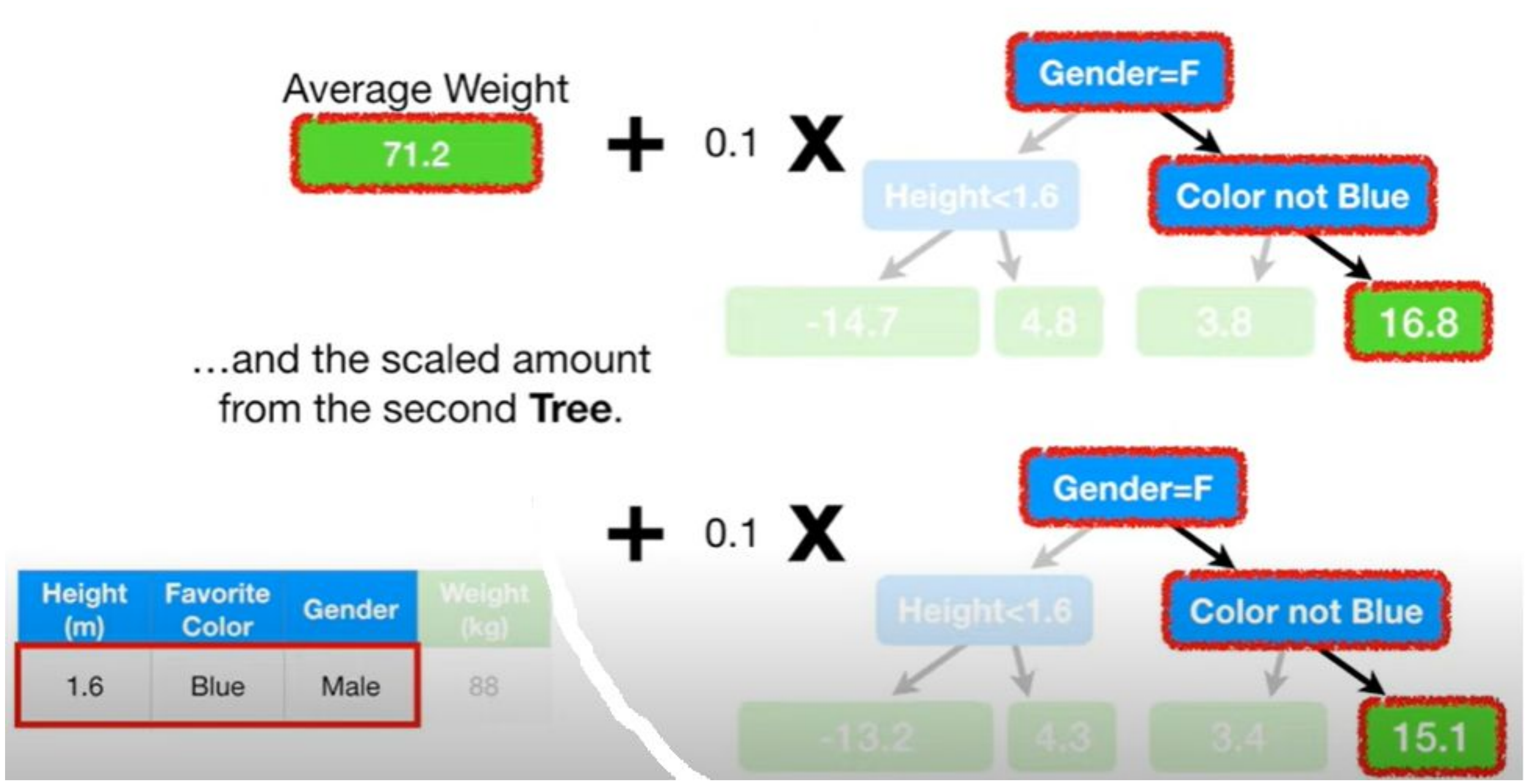
Gradient Boosting



Just like before, since multiple samples ended up in these leaves, we just replace the **Residuals** with their averages.



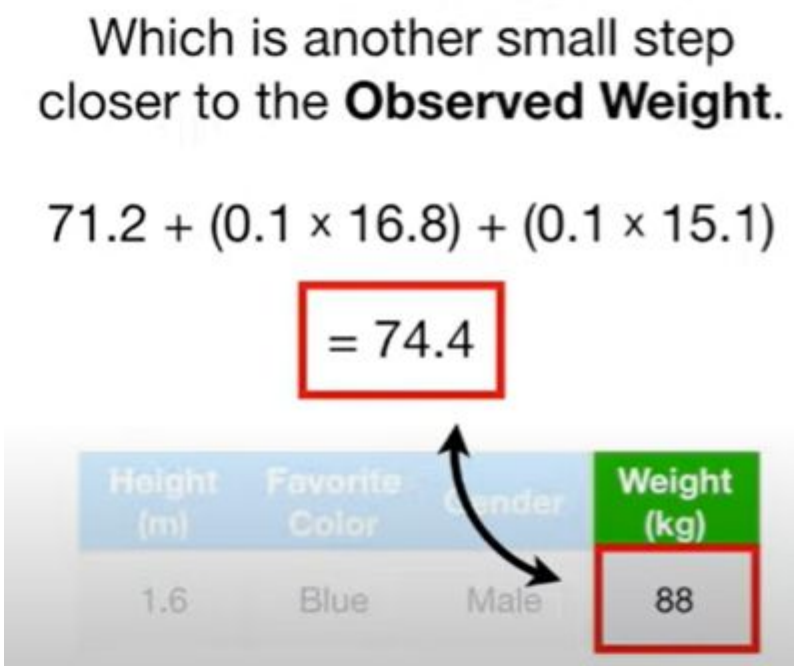
Gradient Boosting



…and the scaled amount from the second **Tree**.

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

Gradient Boosting



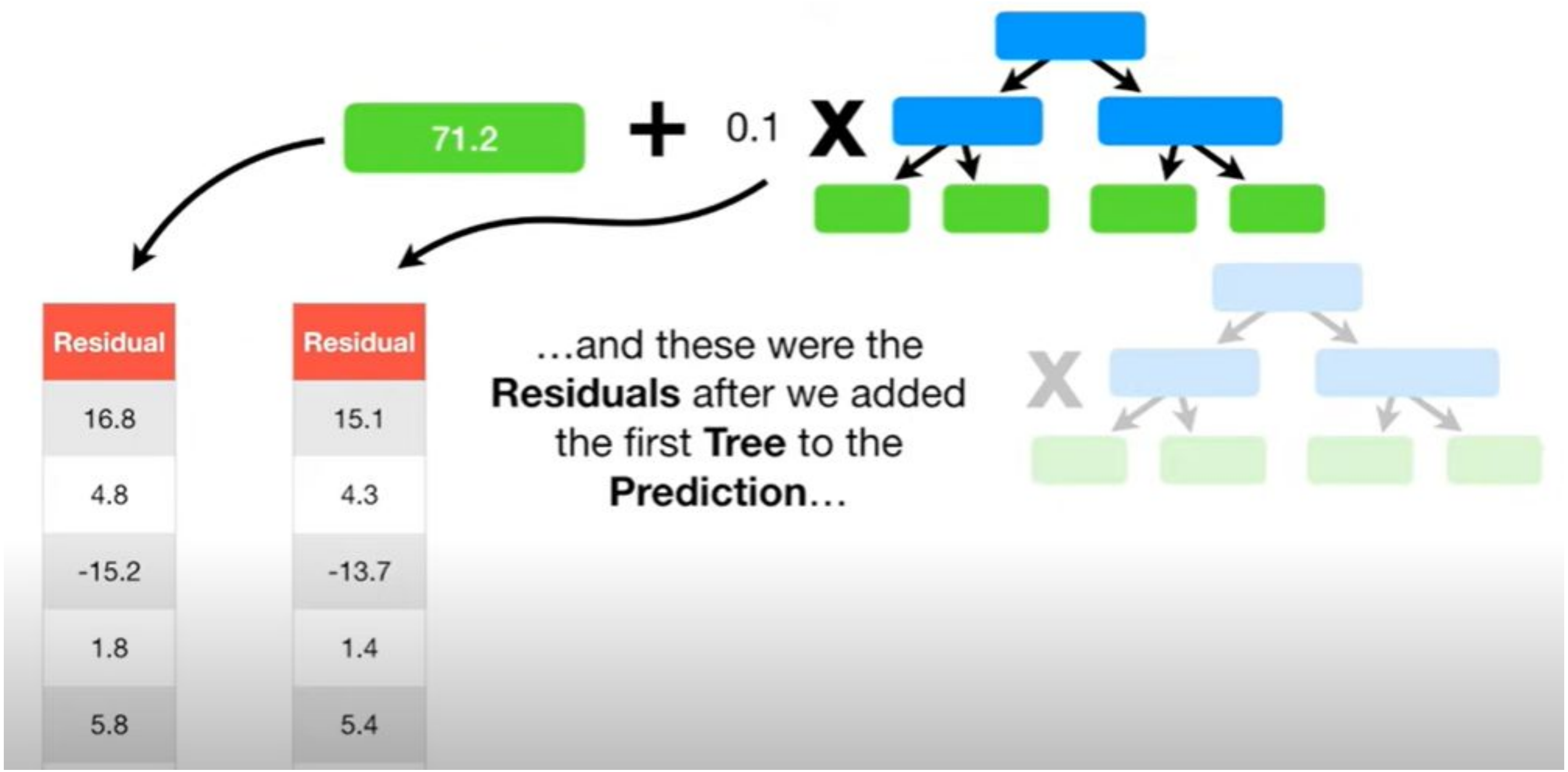
Which is another small step closer to the **Observed Weight**.

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

$$\boxed{= 74.4}$$

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

Gradient Boosting

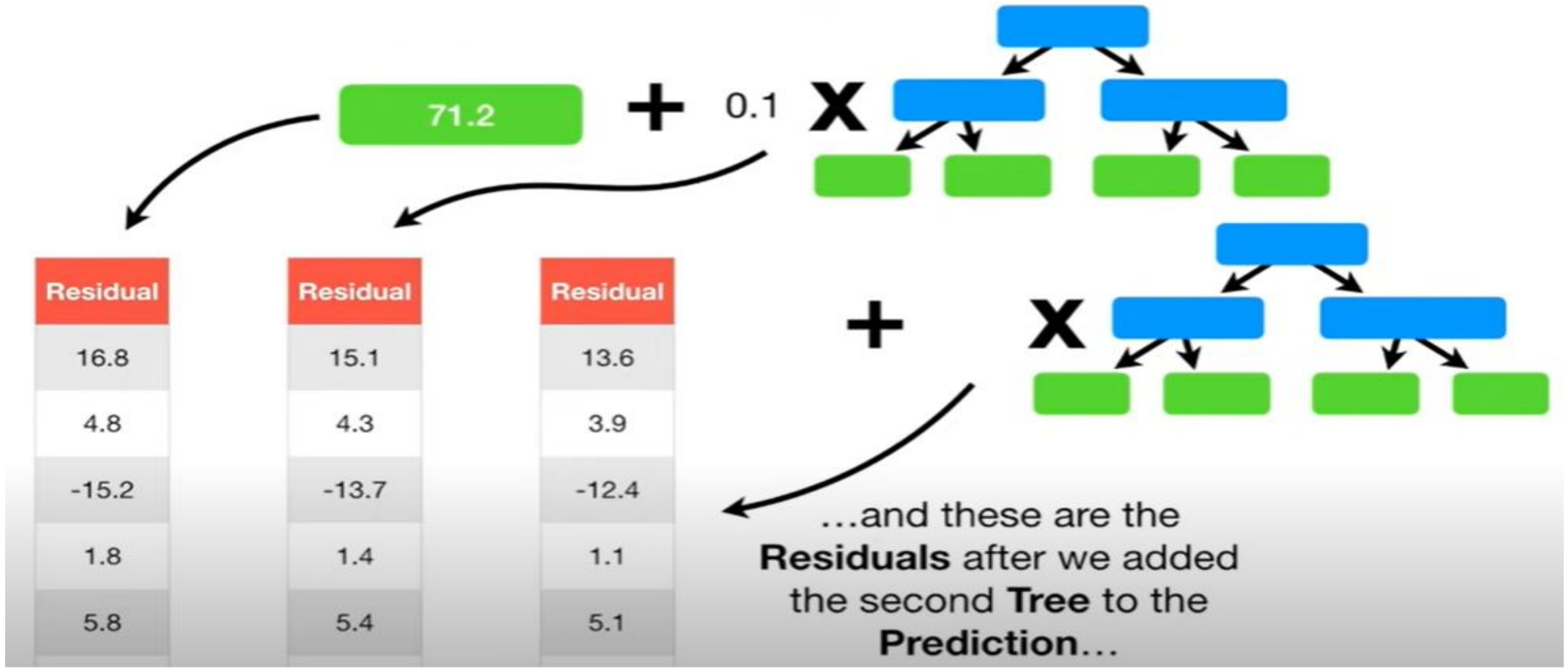


Residual
16.8
4.8
-15.2
1.8
5.8

Residual
15.1
4.3
-13.7
1.4
5.4

…and these were the **Residuals** after we added the first **Tree** to the **Prediction**…

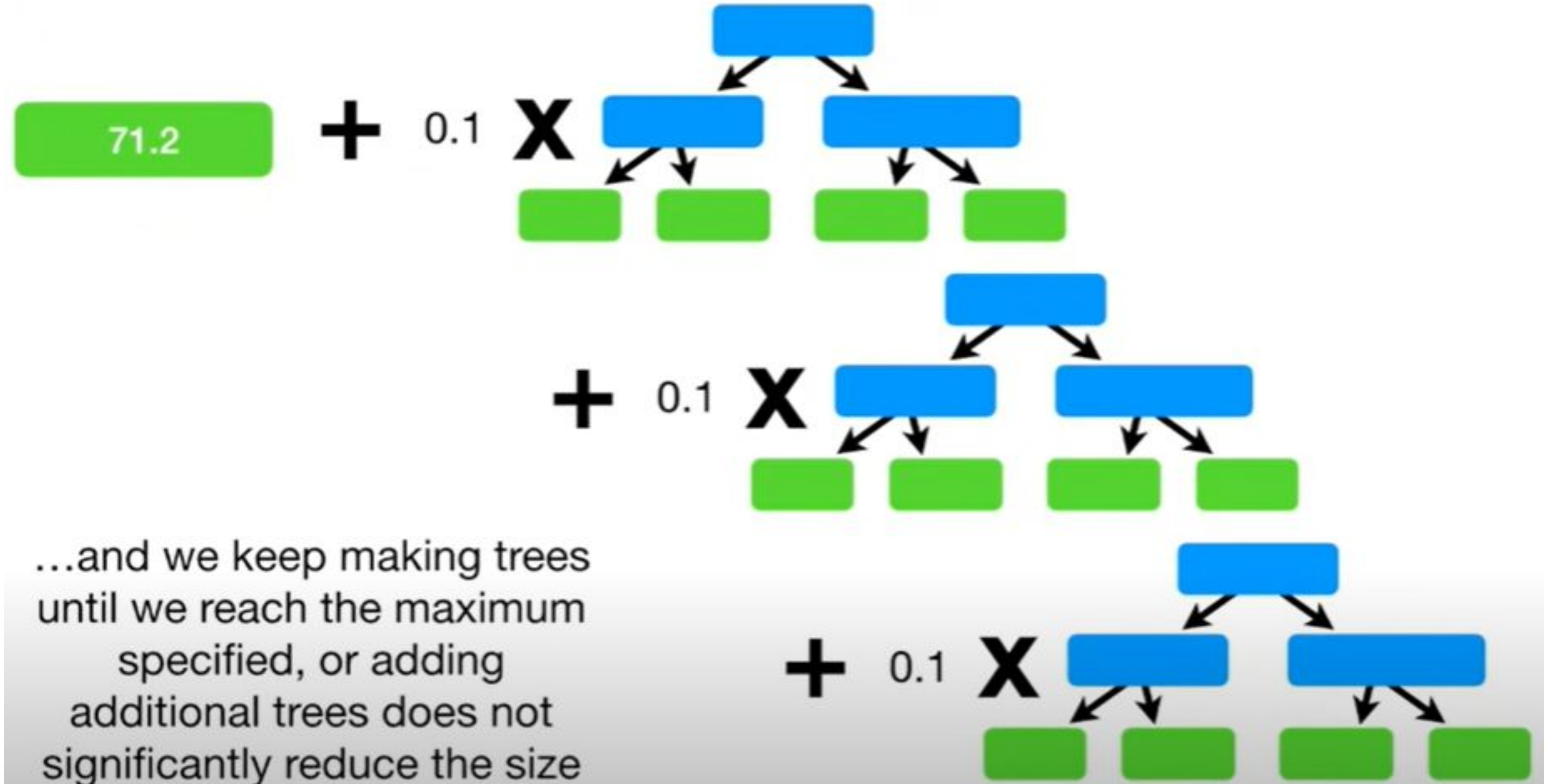
Gradient Boosting



Residual	Residual	Residual
16.8	15.1	13.6
4.8	4.3	3.9
-15.2	-13.7	-12.4
1.8	1.4	1.1
5.8	5.4	5.1

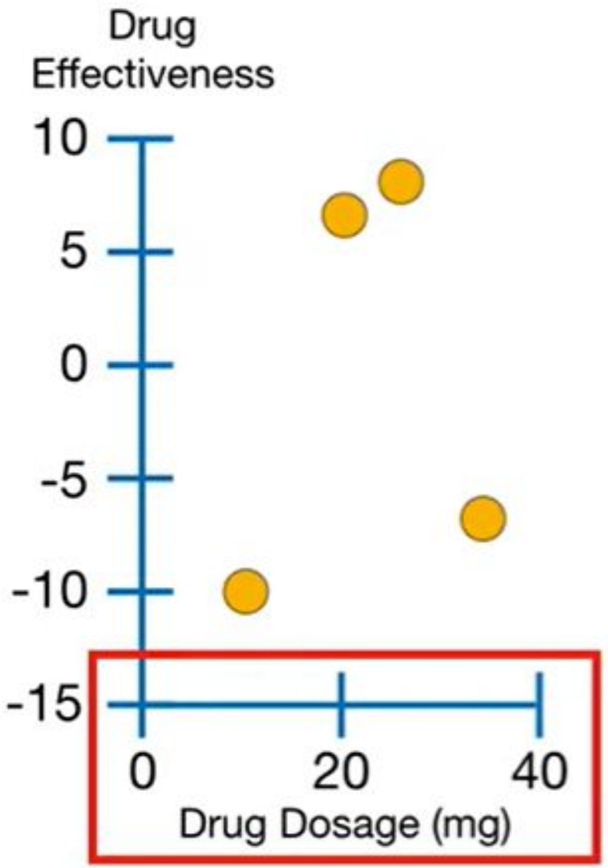
…and these are the **Residuals** after we added the second **Tree** to the **Prediction**…

Each time we add a tree to the **Prediction**, the **Residuals** get smaller.



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

XGBoost



On the ***x*-axis**, we have different **Drug Dosages**…

XGBoost

Predicted Drug Effectiveness

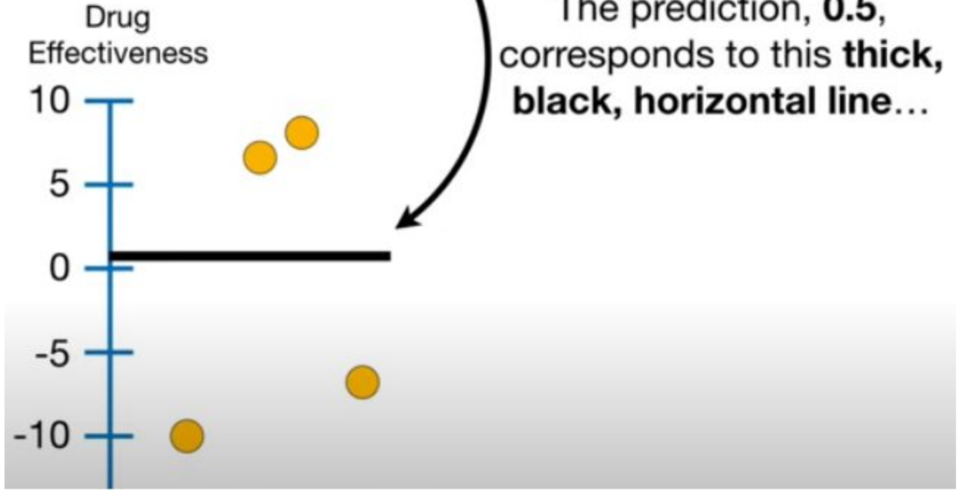
0.5

The very first step in fitting **XGBoost** to the **Training Data** is to make an initial prediction.

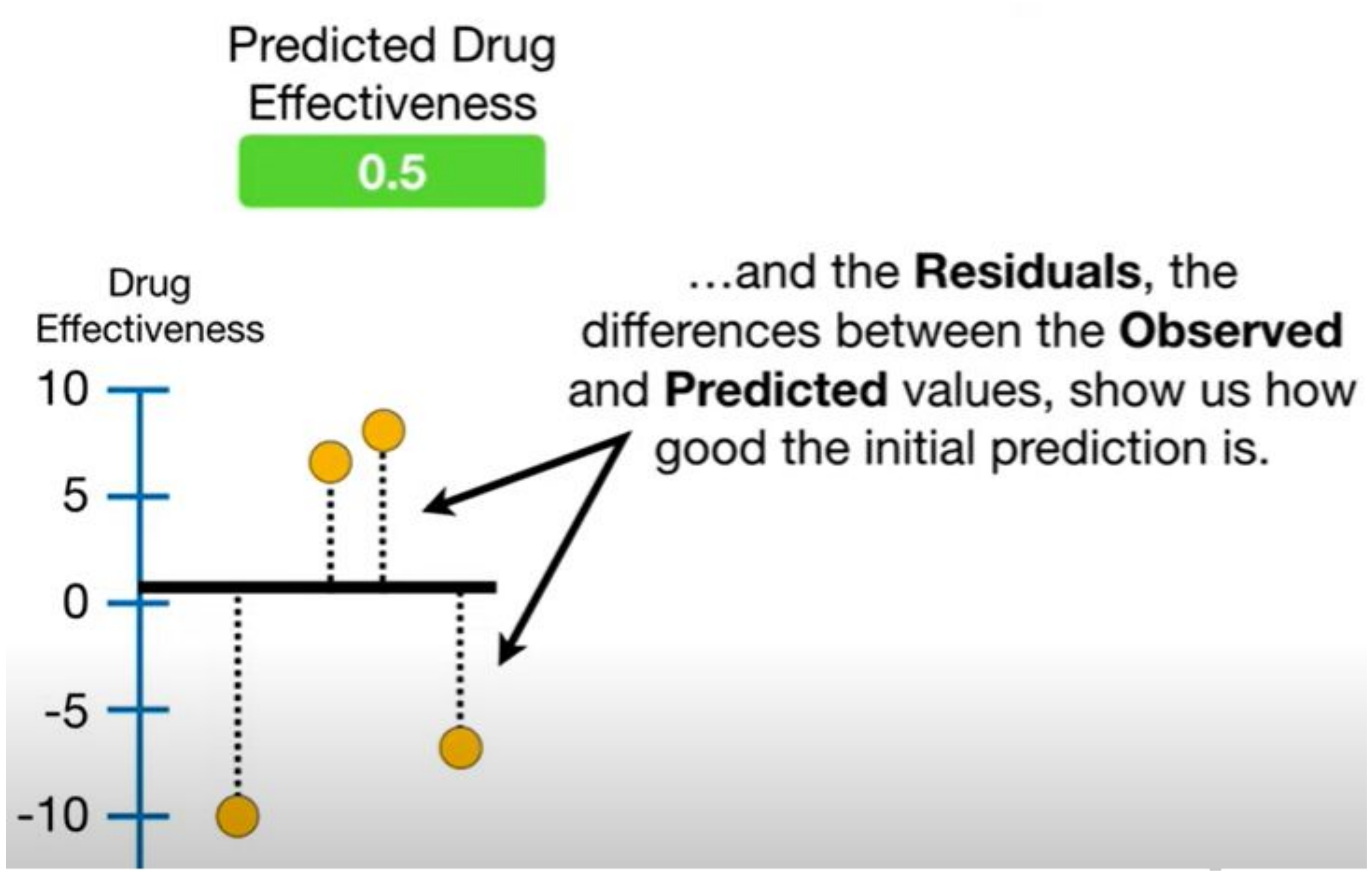
Predicted Drug Effectiveness

0.5

The prediction, **0.5**, corresponds to this **thick, black, horizontal line**…



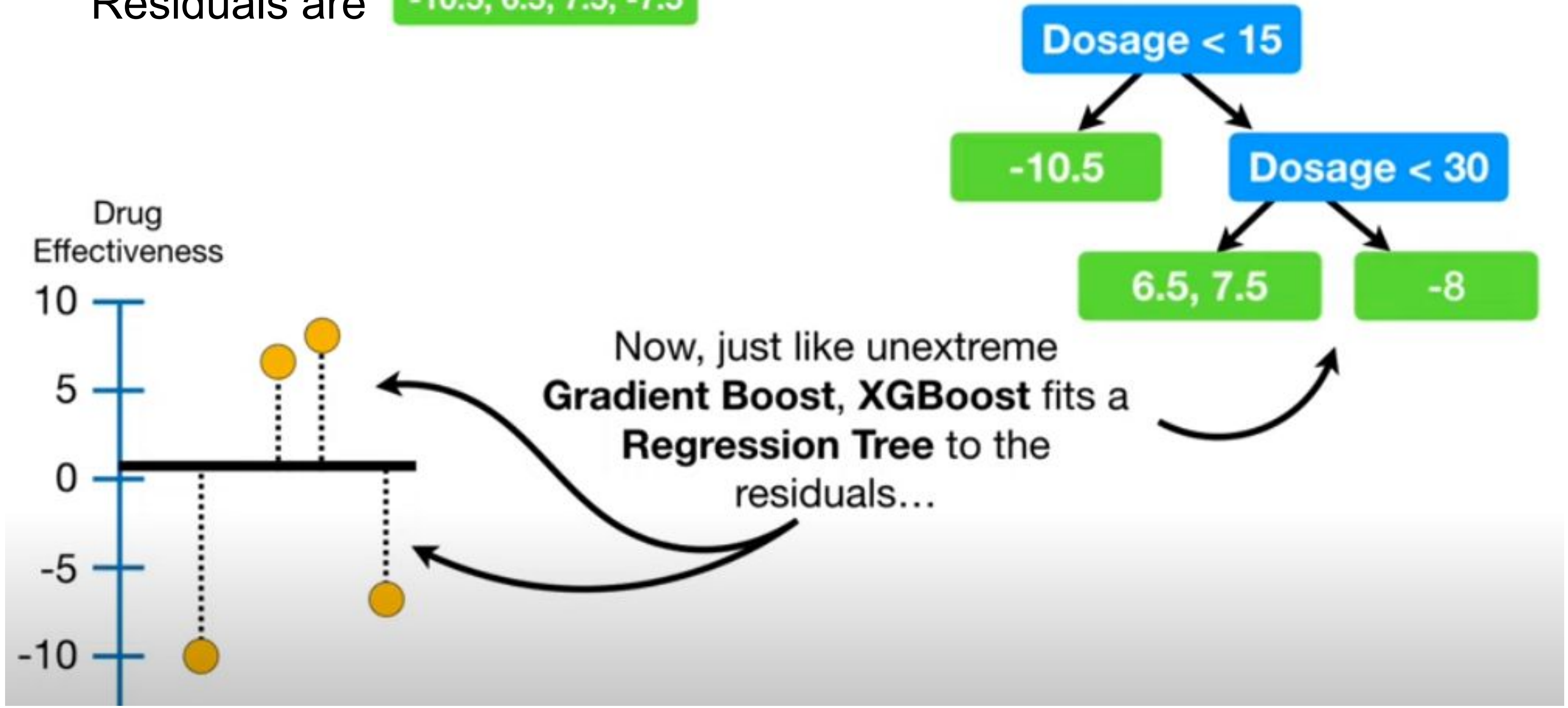
XGBoost



…and the **Residuals**, the differences between the **Observed** and **Predicted** values, show us how good the initial prediction is.

XGBoost

Residuals are -10.5, 6.5, 7.5, -7.5



Now, just like unextreme **Gradient Boost**, **XGBoost** fits a **Regression Tree** to the residuals…

XGBoost

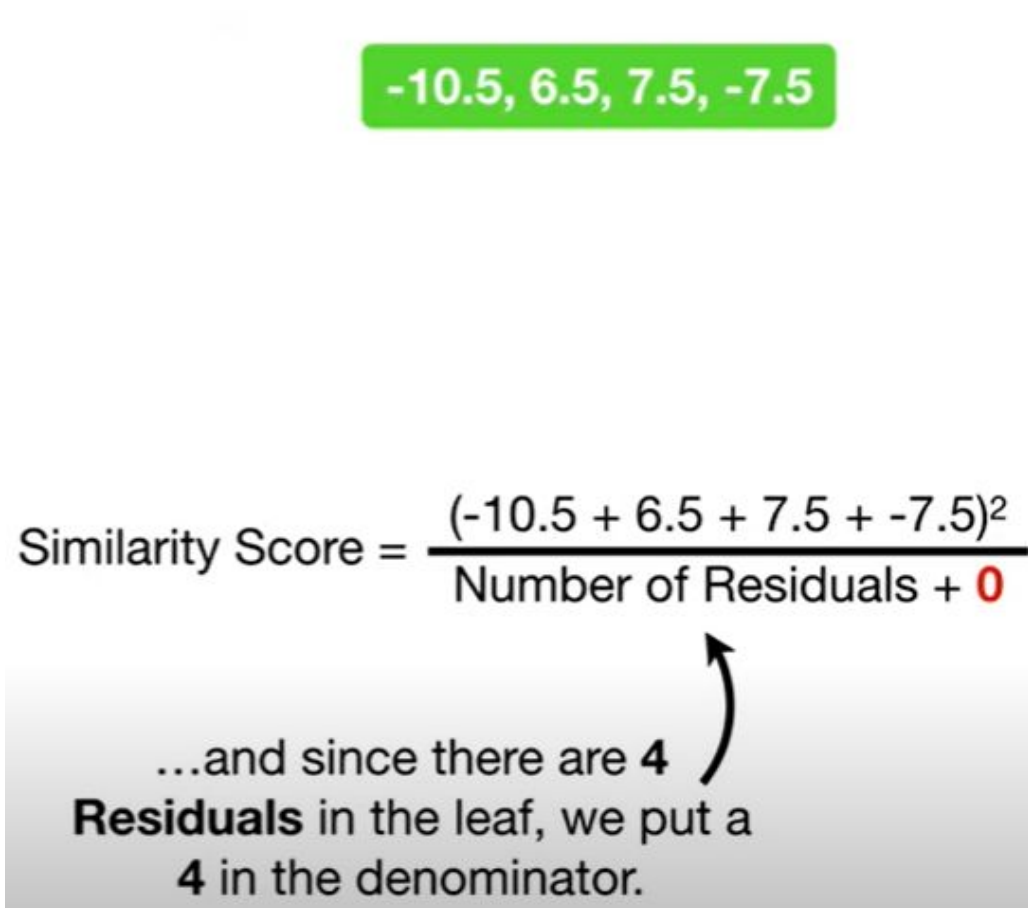
-10.5, 6.5, 7.5, -7.5

Now we calculate a **Quality Score**, or **Similarity Score**, for the **Residuals**.

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 0}$$

For now, let $\lambda = 0$.

XGBoost



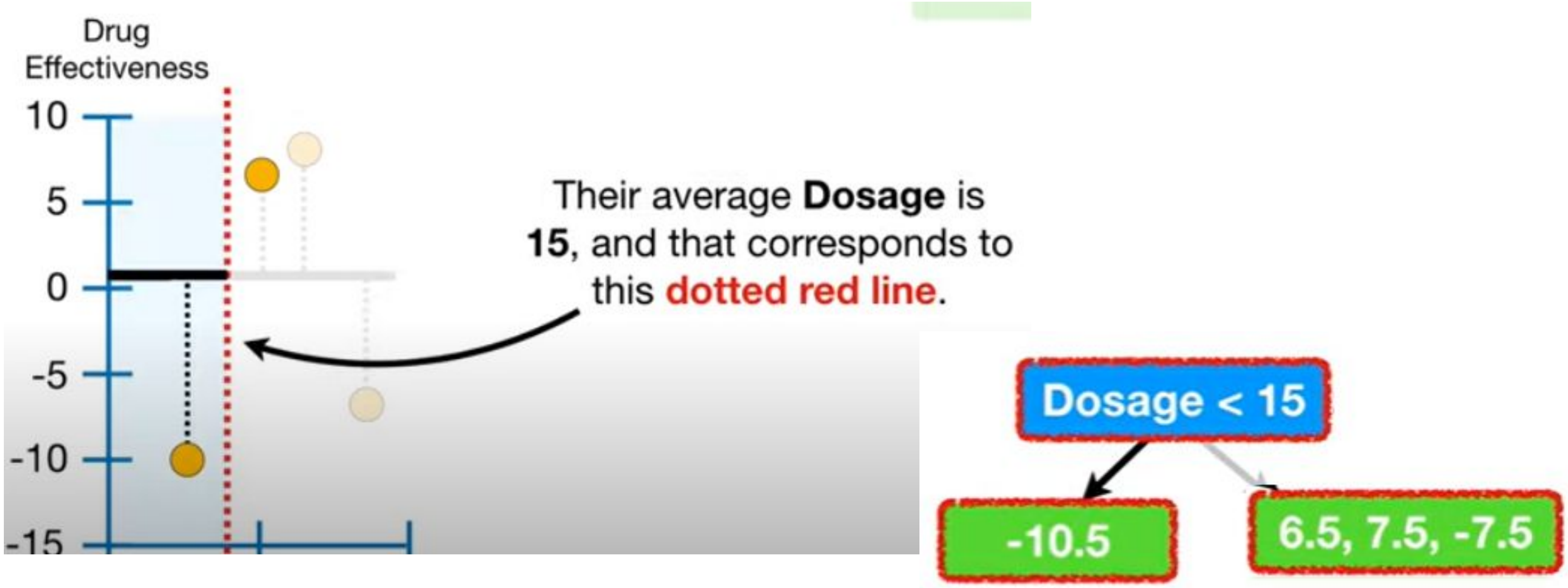
-10.5, 6.5, 7.5, -7.5

$$\text{Similarity Score} = \frac{(-10.5 + 6.5 + 7.5 + -7.5)^2}{\text{Number of Residuals} + \mathbf{0}}$$

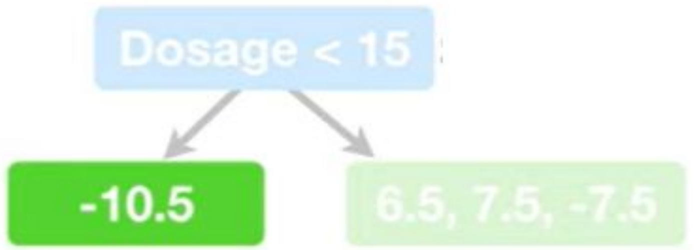
…and since there are **4** **Residuals** in the leaf, we put a **4** in the denominator.

XGBoost

Consider first two observations with lowest dosages.



XGBoost



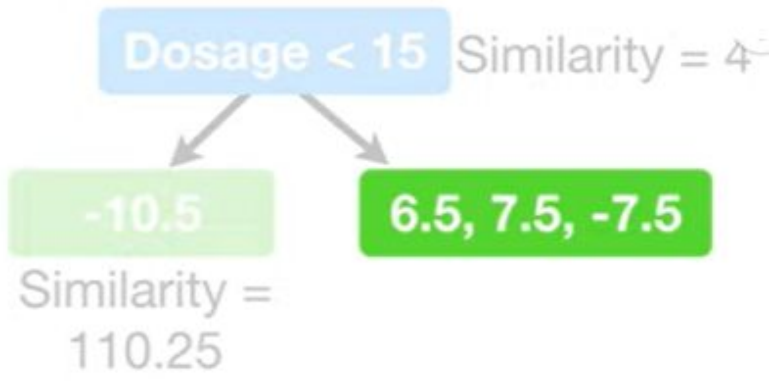
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

$$\text{Similarity Score} = \frac{-10.5^2}{1 + \lambda}$$

Now we calculate the **Similarity Score** for the leaf on the left…

$$\text{Similarity Score} = \frac{-10.5^2}{1 + 0} \boxed{= 110.25}$$

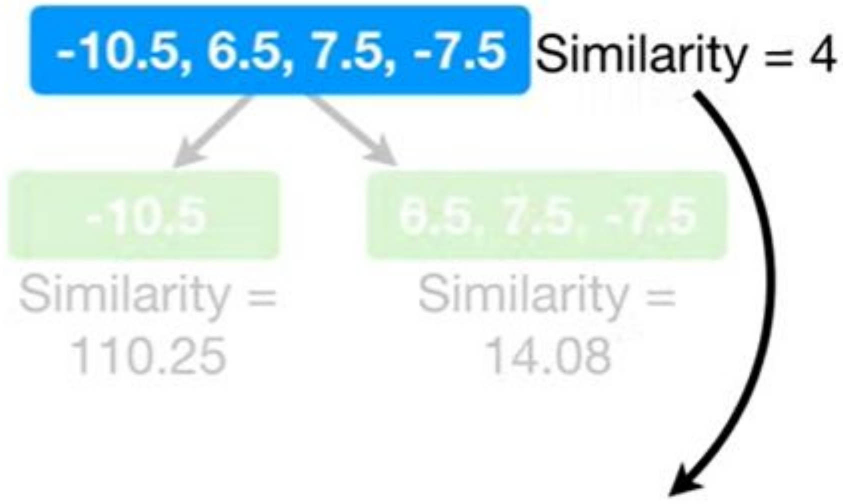
XGBoost



$$\text{Similarity Score} = \frac{6.5^2}{3 + 0} \boxed{= 14.08}$$

Thus, the **Similarity Score** for the **Residuals** in the leaf on the right = **14.08**.

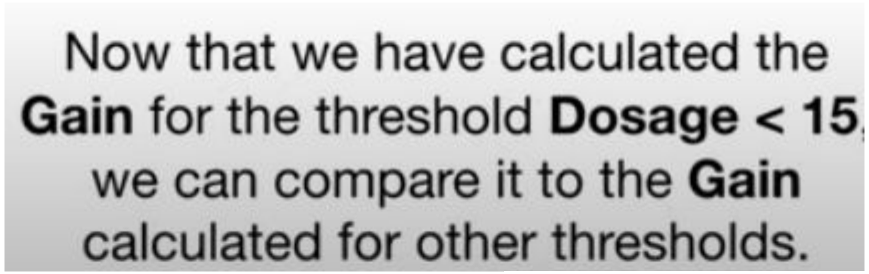
XGBoost



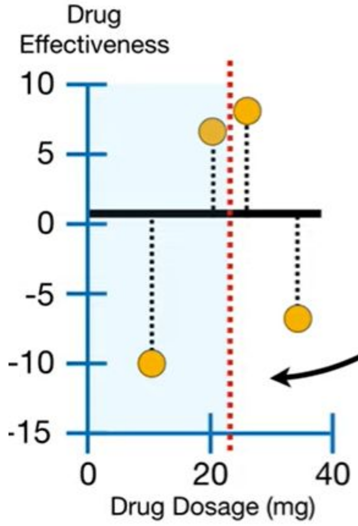
$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

$$\text{Gain} = 110.25 + 14.08 - 4 = 120.33$$

XGBoost

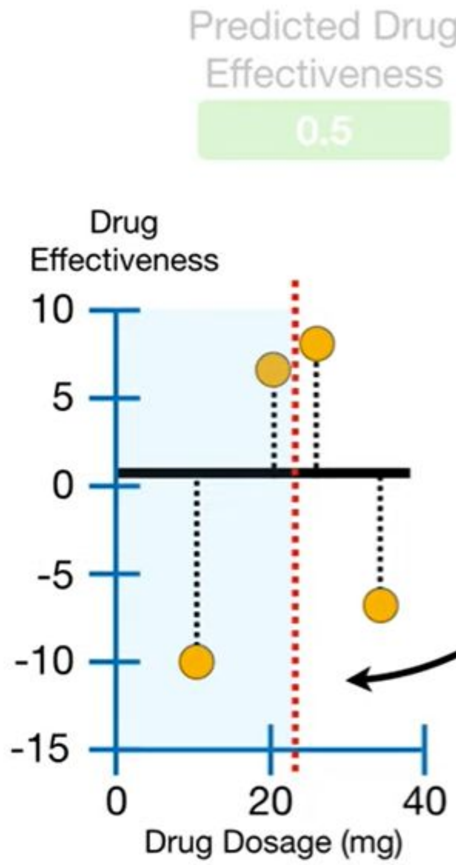


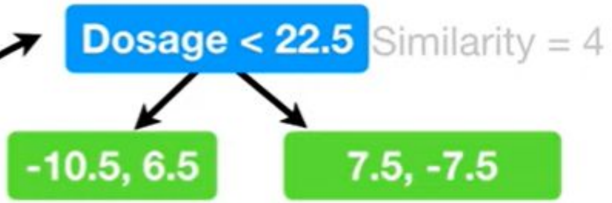
Now that we have calculated the **Gain** for the threshold **Dosage < 15** we can compare it to the **Gain** calculated for other thresholds.



So we shift the threshold over so that it is the average of the next two observations...

XGBoost



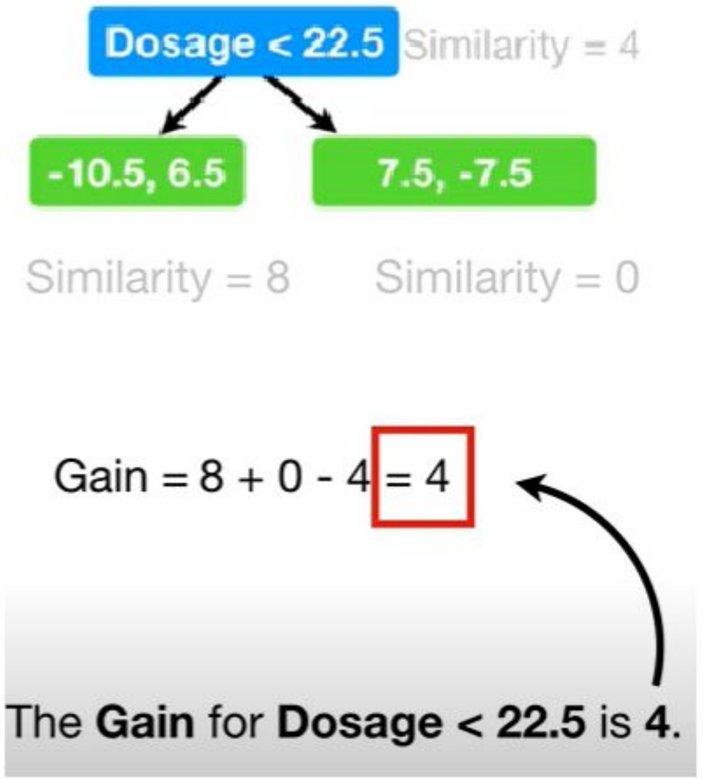


…and build a simple tree that divides the observations using the new threshold, **Dosage < 22.5**.

XGBoost

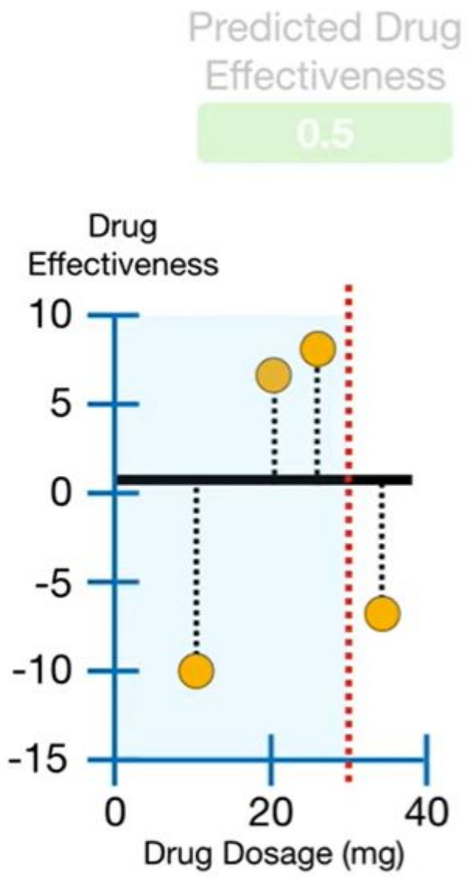
Gain = 8 + 0 - 4 = 4

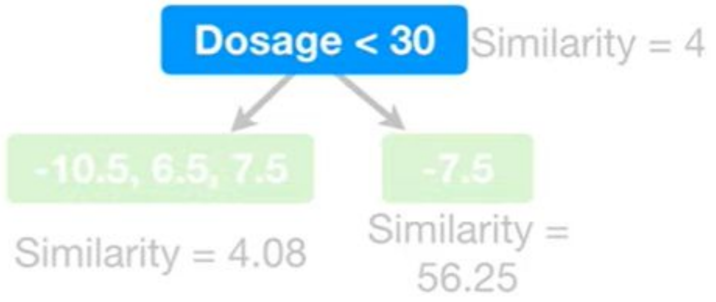
Since the **Gain** for **Dosage < 22.5** (**Gain = 4**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the **Residuals** into clusters of similar values.



The **Gain** for **Dosage < 22.5** is **4**.

XGBoost



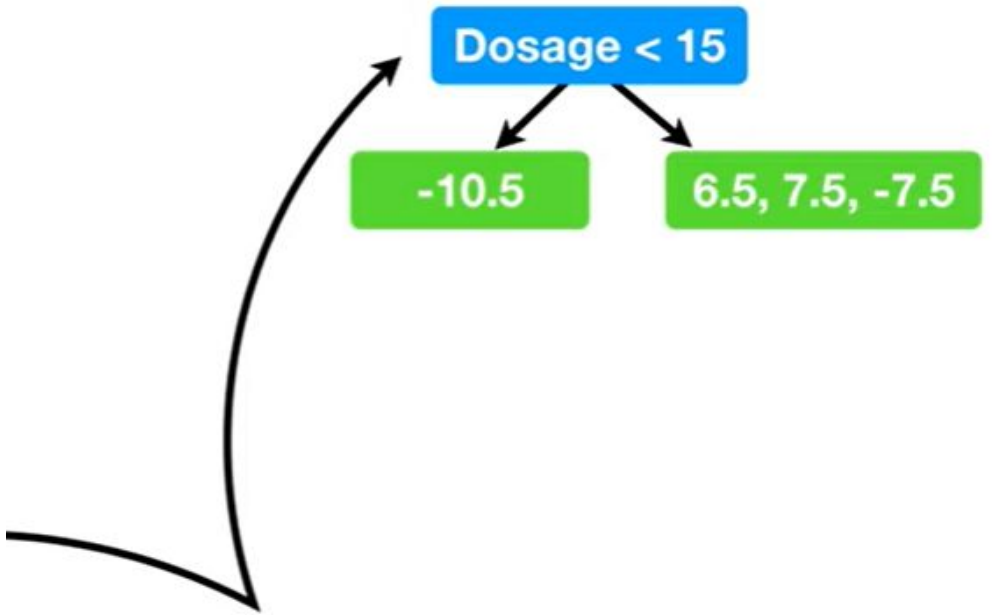


$$\text{Gain} = 4.08 + 56.25 - 4 \boxed{= 56.33}$$

The **Gain** for **Dosage < 30 = 56.33**

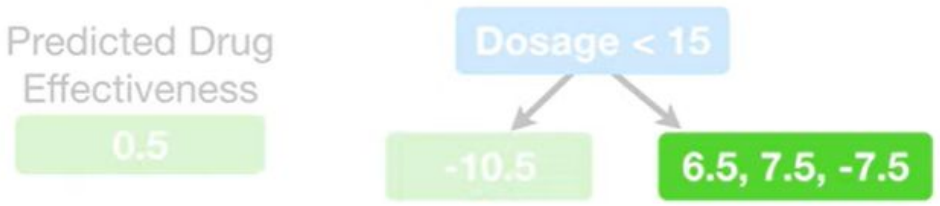
Again, since the **Gain** for **Dosage < 30** (**Gain = 56.33**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the observations.

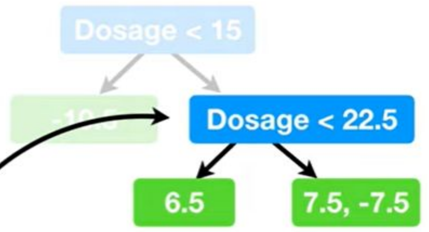
XGBoost



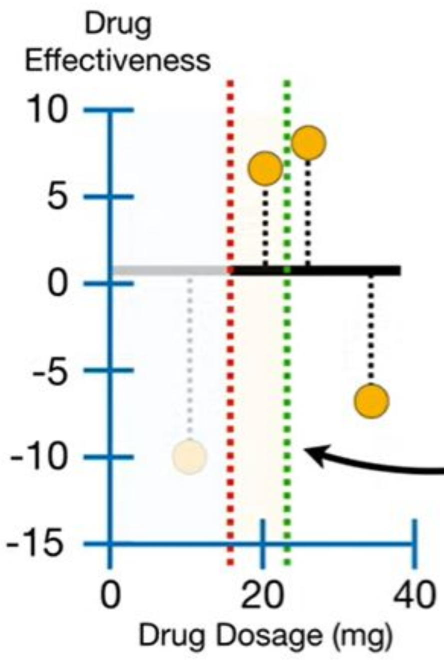
…and we will use the threshold that gave us the largest **Gain**, **Dosage < 15**, for the first branch in the tree.

XGBoost





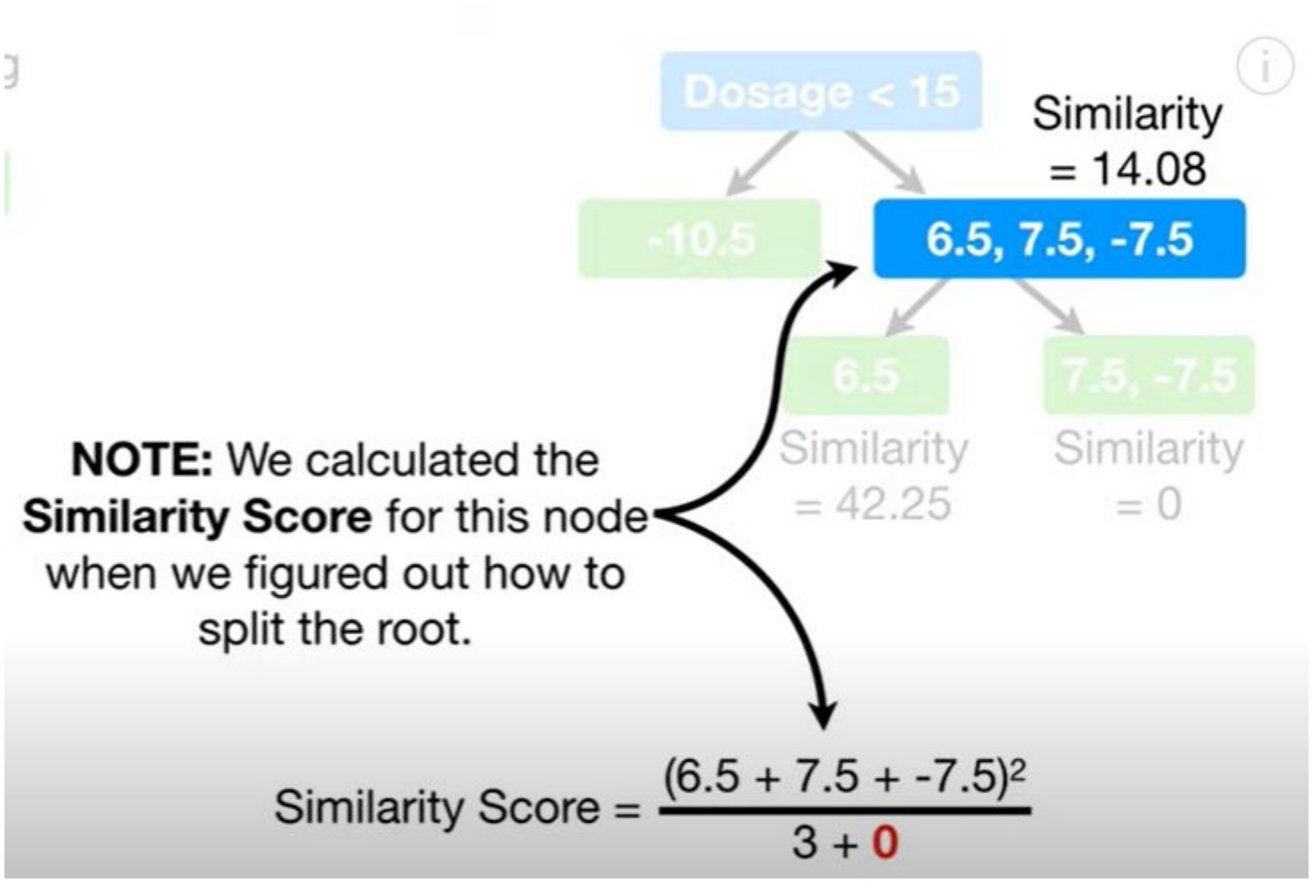
So the first threshold that we try is **Dosage < 22.5**.



…and their average **Dosage** is **22.5**, which corresponds to this **dotted green line**.

And we get **Gain = 28.17** for when the threshold is **Dosage < 22.5**.

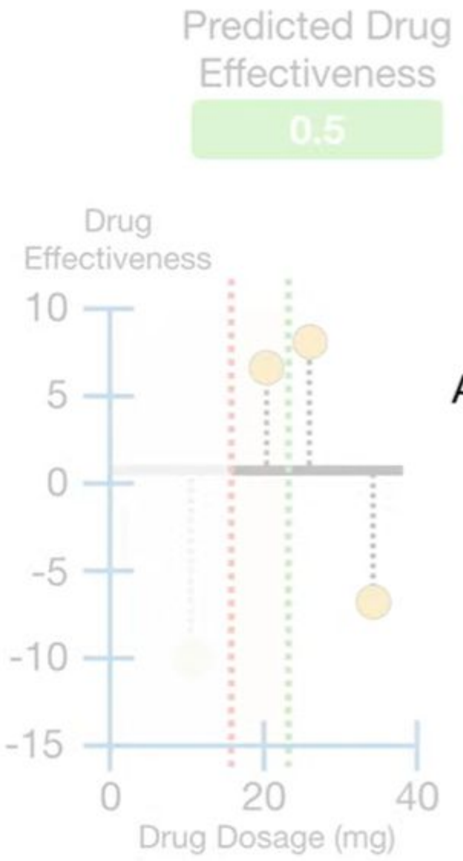
XGBoost

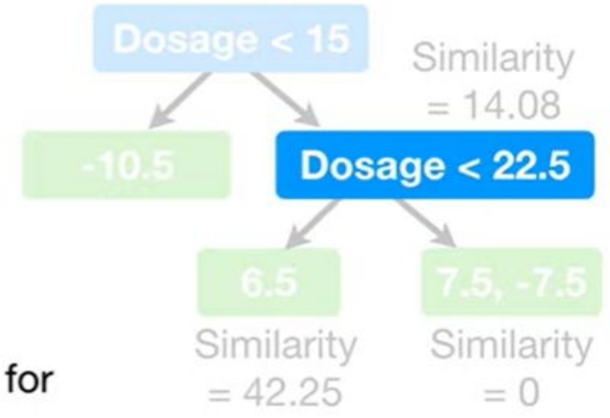


NOTE: We calculated the **Similarity Score** for this node when we figured out how to split the root.

$$\text{Similarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{3 + \mathbf{0}}$$

XGBoost

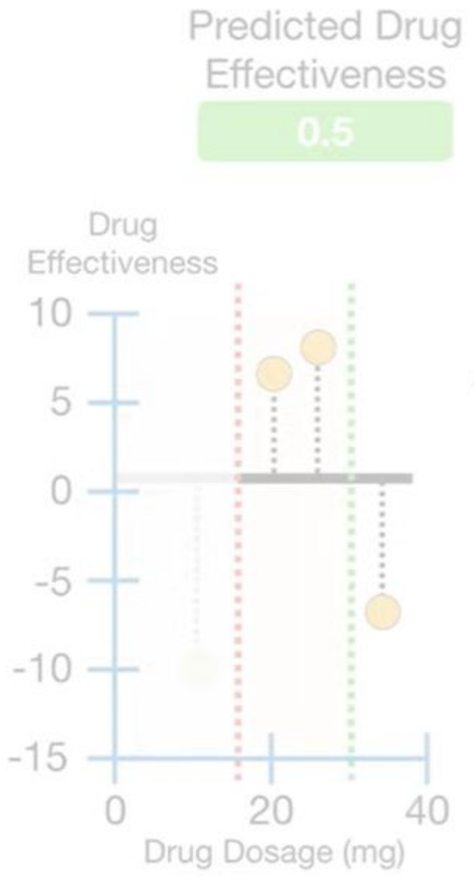


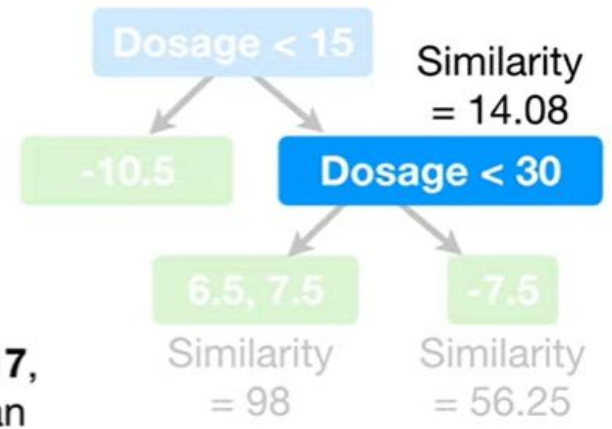


And we get **Gain = 28.17** for when the threshold is **Dosage < 22.5**.

Gain = 42.25 + 0 - 14.08 = 28.17

XGBoost

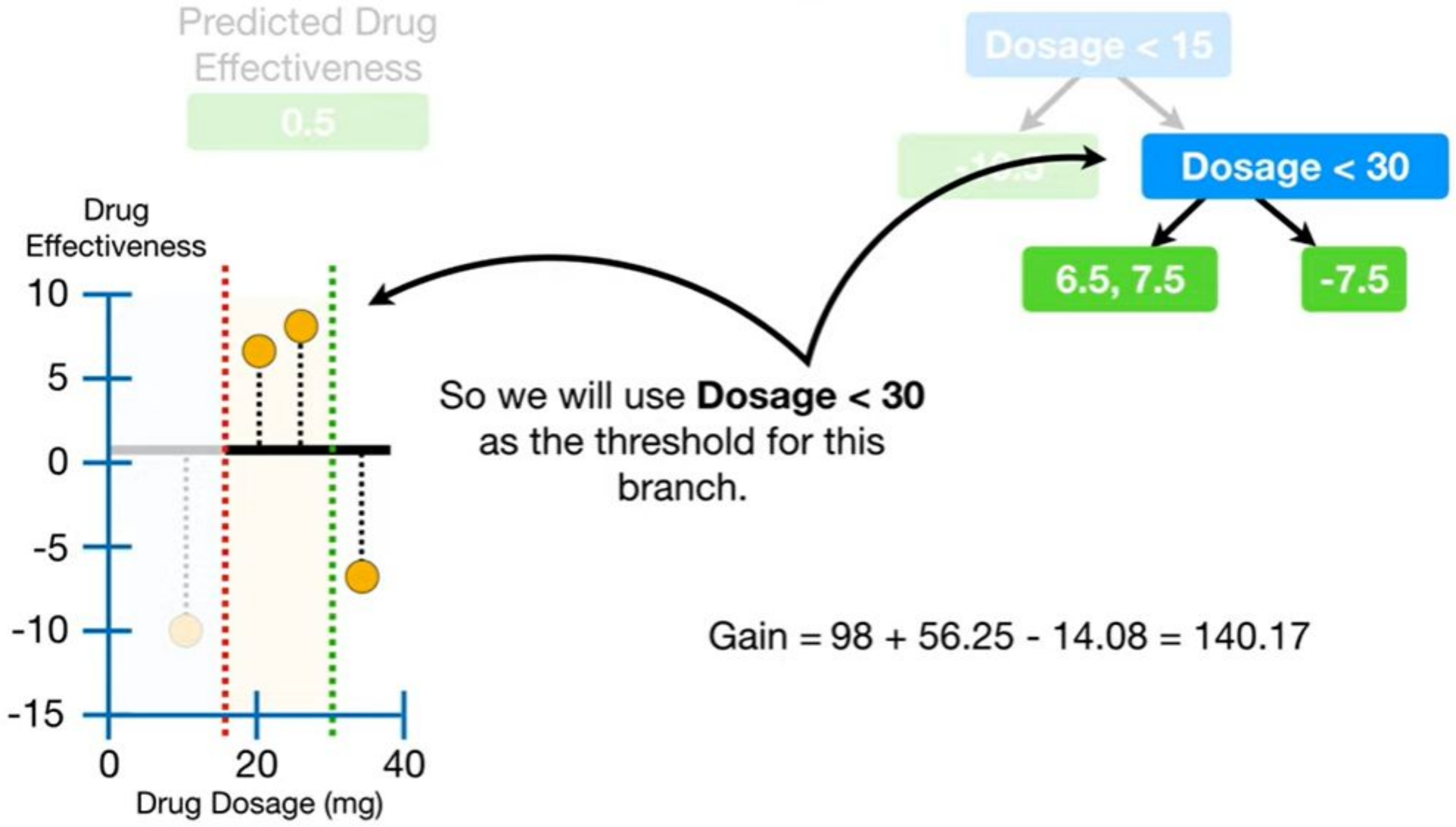




And we get **Gain = 140.17**, which is much larger than **28.17**, when the threshold was **Dosage < 22.5**.

$$\text{Gain} = 98 + 56.25 - 14.08 = 140.17$$

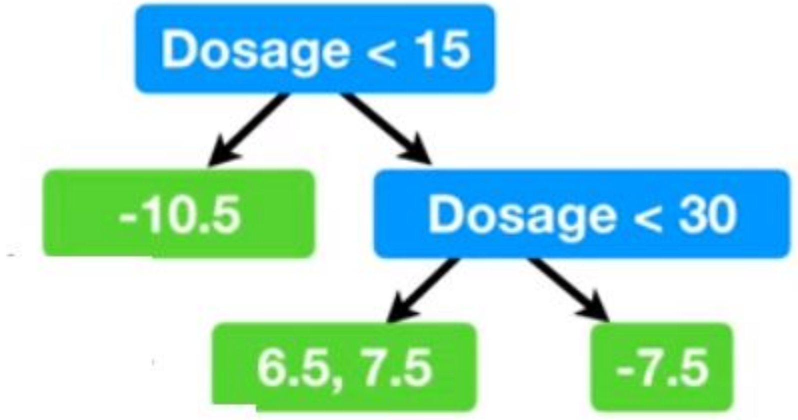
XGBoost



So we will use **Dosage < 30** as the threshold for this branch.

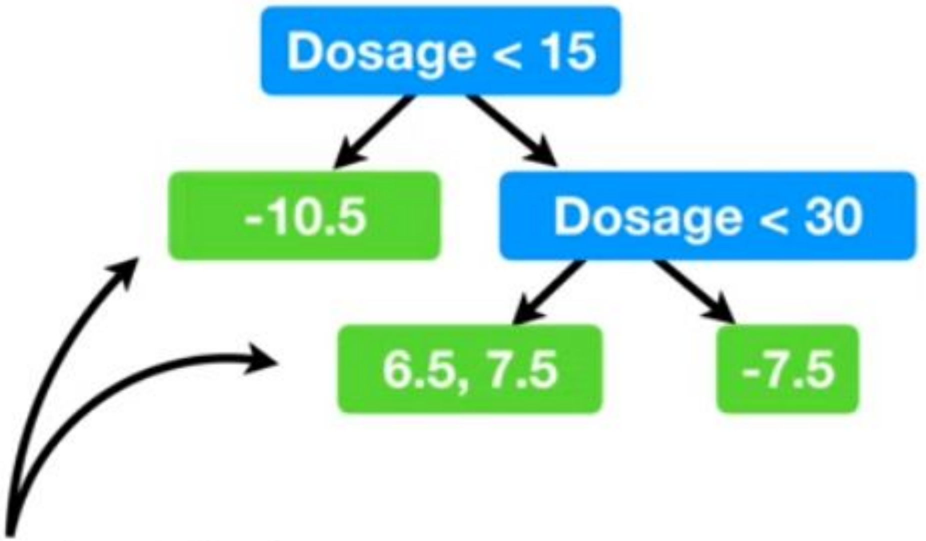
Gain = 98 + 56.25 - 14.08 = 140.17

XGBoost



This tree can be further splitted if required

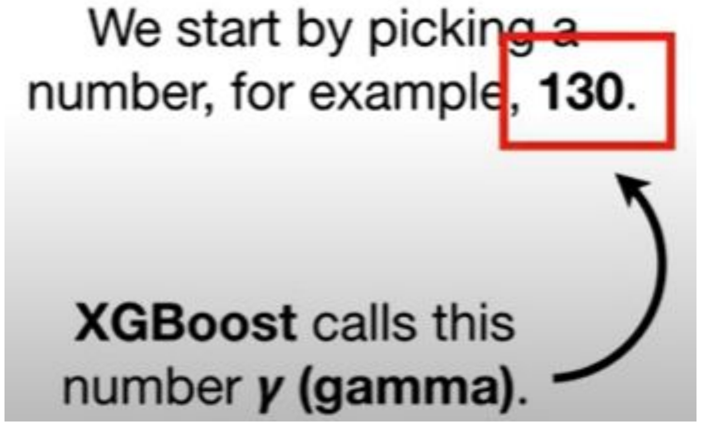
XGBoost



Now we need to talk about how to **Prune** this tree.

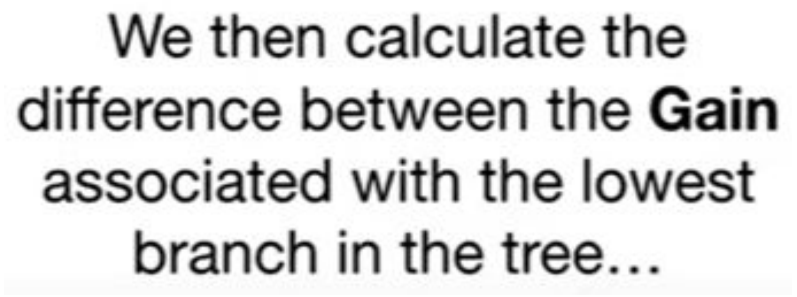
We **Prune** an **XGBoost Tree** based on its **Gain** values.

XGBoost



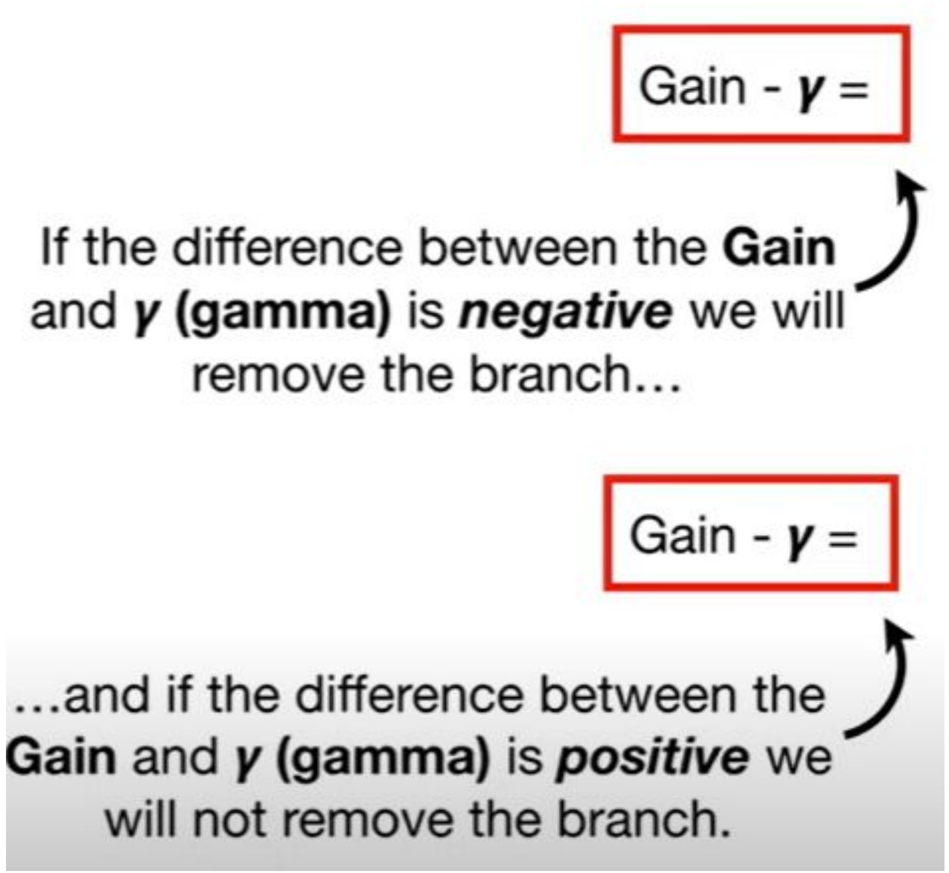
We start by picking a number, for example, **130**.

XGBoost calls this number ***γ* (gamma)**.



We then calculate the difference between the **Gain** associated with the lowest branch in the tree…

XGBoost



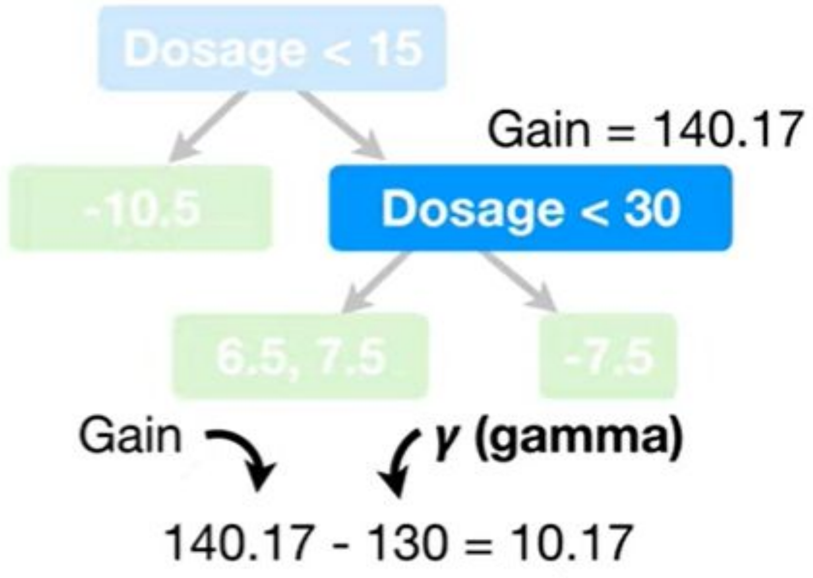
$\boxed{\text{Gain} - \boldsymbol{\gamma} =}$

If the difference between the **Gain** and ***γ* (gamma)** is ***negative*** we will remove the branch…

$\boxed{\text{Gain} - \boldsymbol{\gamma} =}$

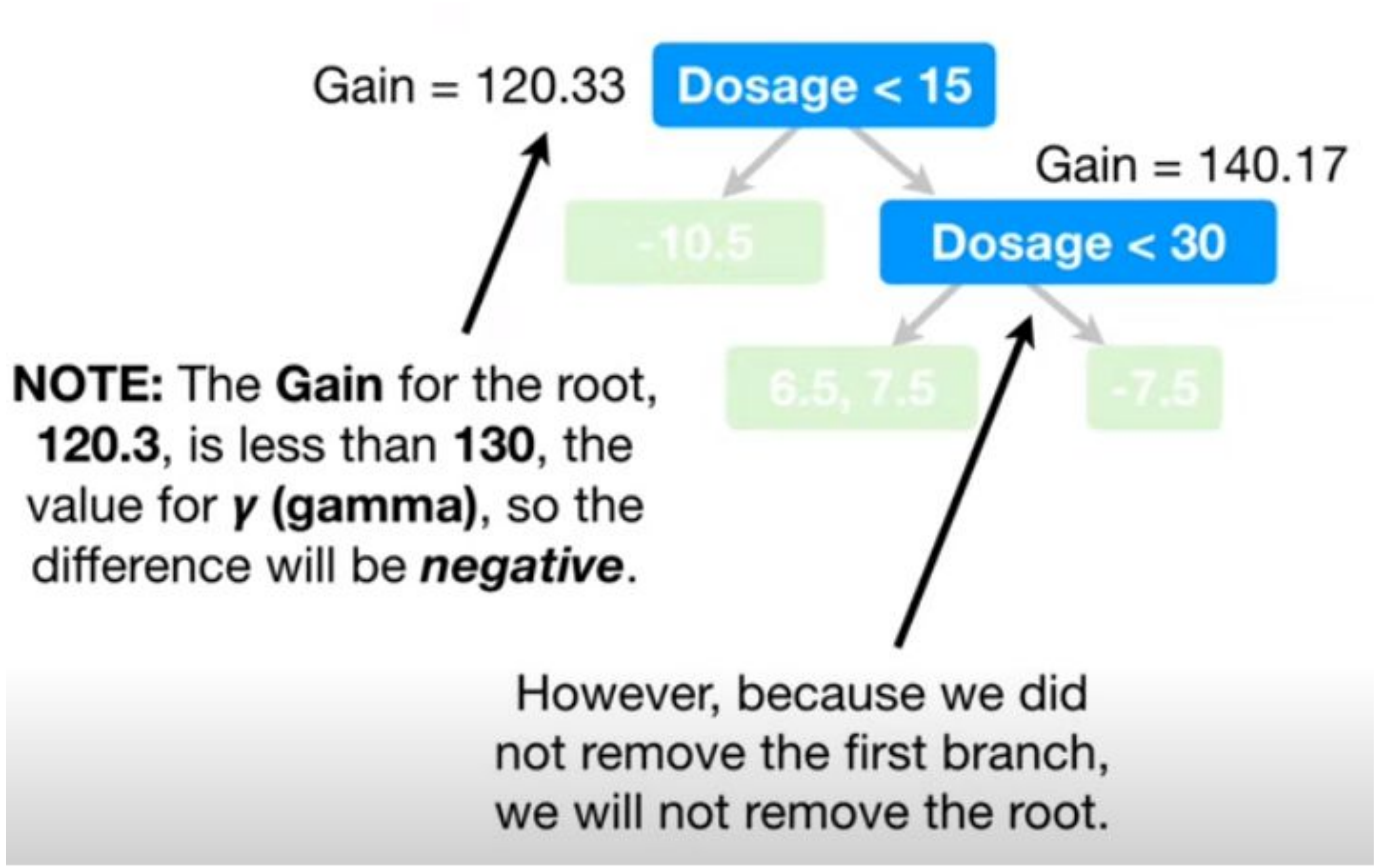
…and if the difference between the **Gain** and ***γ* (gamma)** is ***positive*** we will not remove the branch.

XGBoost



…we get a ***positive*** number, so we will not remove this branch and we are done pruning.

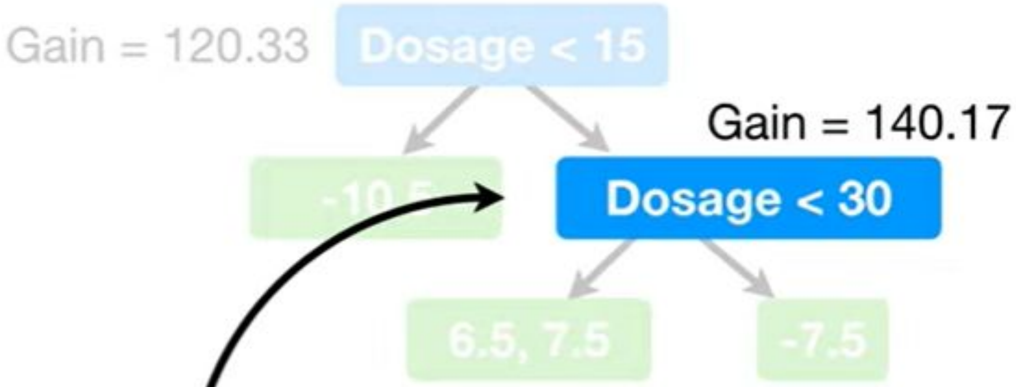
XGBoost



NOTE: The **Gain** for the root, **120.3**, is less than **130**, the value for ***γ* (gamma)**, so the difference will be ***negative***.

However, because we did not remove the first branch, we will not remove the root.

XGBoost



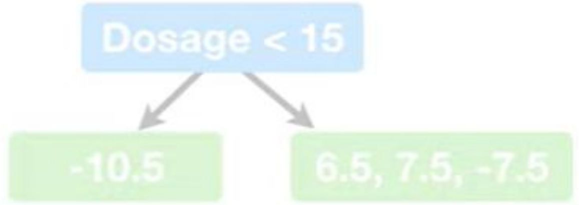
In contrast, if we set ***γ*** = **150**,
then we would remove this
branch because…
140.17 - 150 = a negative number.

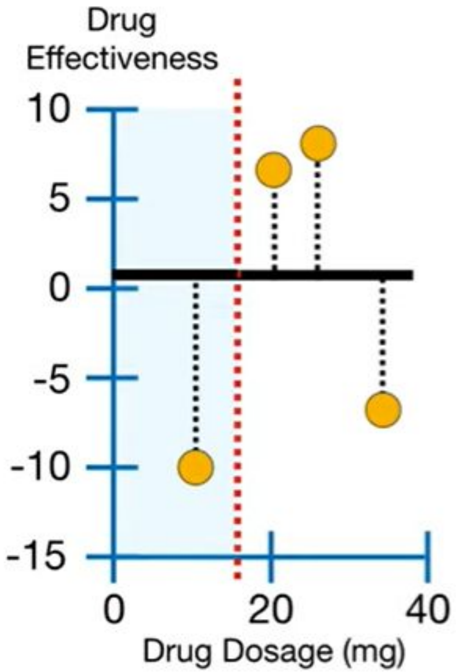
Gain

***γ* (gamma)**

XGBoost

Predicted Drug Effectiveness
0.5



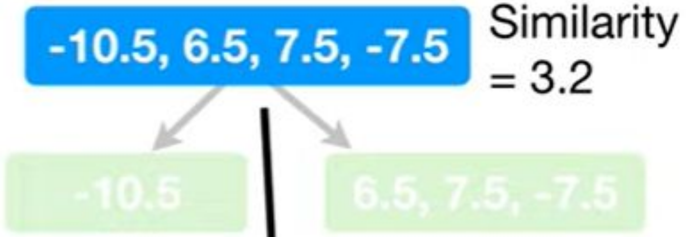


$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

…only this time, when we calculate **Similarity Scores**, we will set **λ (lambda) = 1**.

Remember **λ (lambda)** is a **Regularization Parameter**, which means that it is intended to reduce the prediction's sensitivity to individual observations.

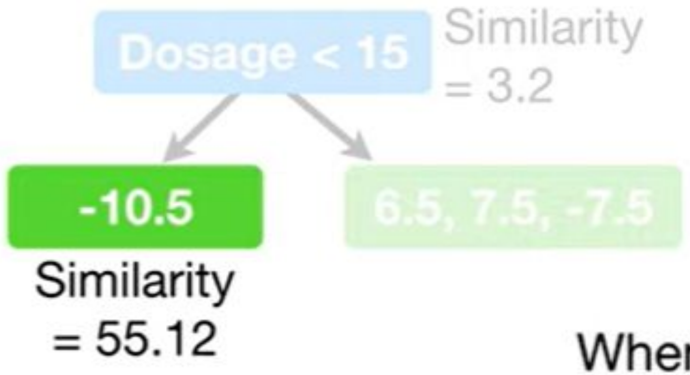
XGBoost



$$\text{Similarity Score} = \frac{(-10.5 + 6.5 + 7.5 + -7.5)^2}{4 + 1} \boxed{= 3.2}$$

…**3.2**, which is **8/10s** of what we got when **λ = 0**.

XGBoost

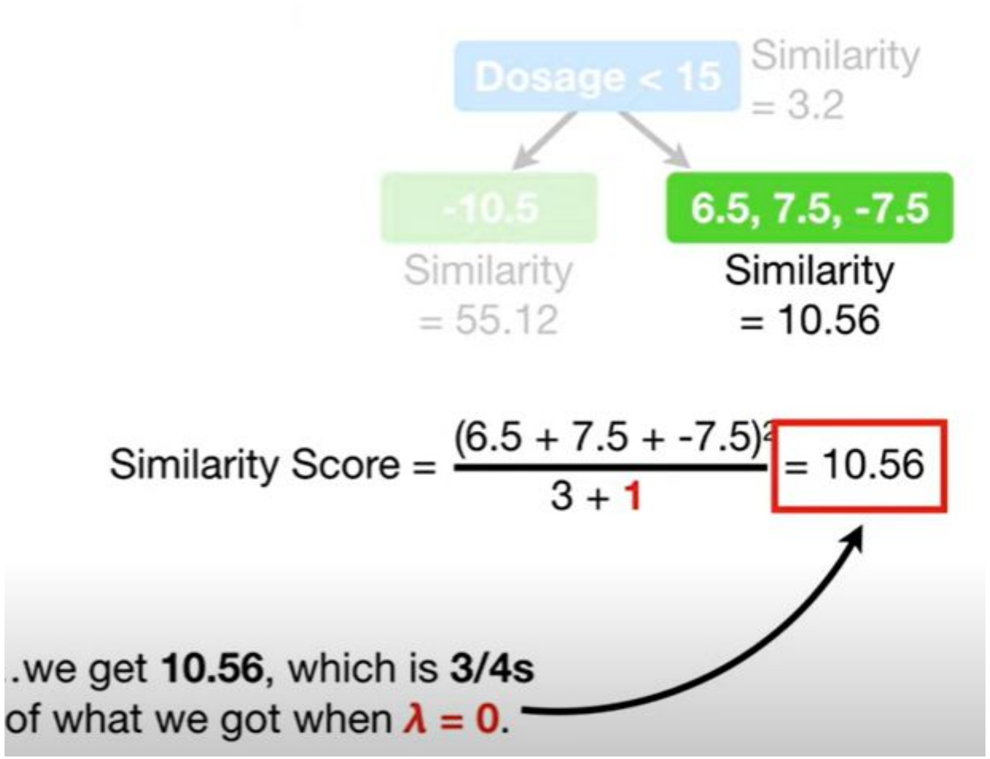


When we calculate the **Similarity Score** for the leaf on the left…

$$\text{Similarity Score} = \frac{-10.5^2}{1 + 1} \boxed{= 55.12}$$

..we get **55.12**, which is half of what we got when **$\lambda = 0$**.

XGBoost



$$\text{Similarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{3 + 1} \boxed{= 10.56}$$

...we get **10.56**, which is **3/4s** of what we got when **$\lambda = 0$**.

And when we calculate the **Similarity Score** for the leaf on the right…

XGBoost

So, one thing we see is that when $\lambda > 0$, the **Similarity Scores** are smaller…

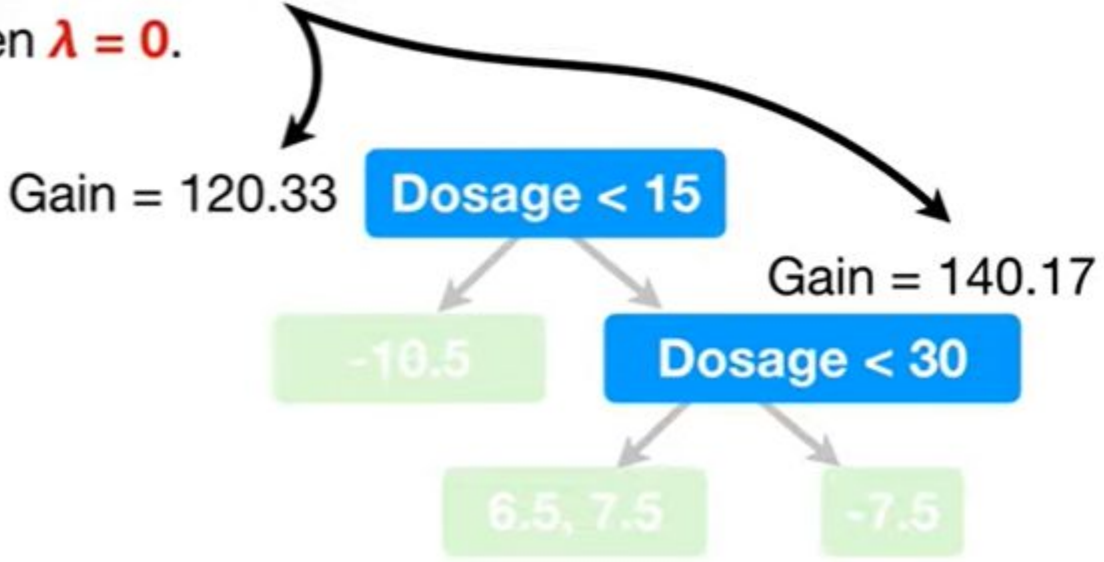
…and the amount of decrease is **inversely proportional** to the number of **Residuals** in the node.

In other words, the leaf on the left had only **1 Residual**, and it had the largest decrease in **Similarity Score**, **50%**.

In contrast, the root had all **4 Residuals** and the smallest decrease, **20%**.

XGBoost

Now, just for comparison, these were the **Gain** values when **$\lambda = 0$**.



XGBoost

When we first talked about pruning trees, we set **γ (gamma) = 130**…

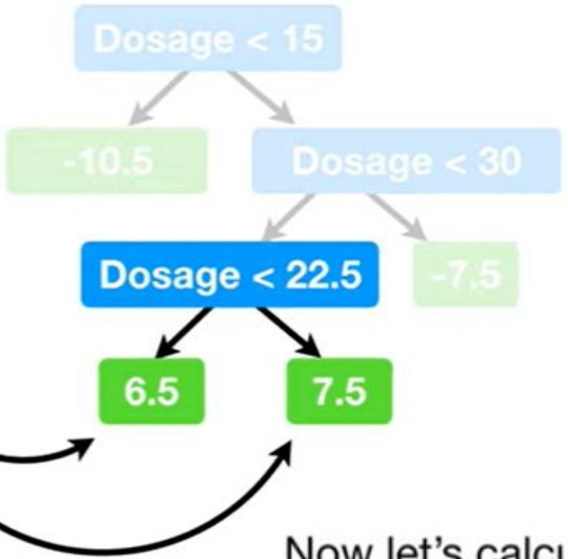
…and because, for the lowest branch in the first tree, **Gain - γ = a positive number**, we did not prune at all.

Now, with **λ (lambda) = 1**, the values for **Gain** are both **< 130**…

…so we would prune the whole tree away.

So when **$\lambda > 0$**, it is easier to prune leaves because the values for **Gain** are smaller.

XGBoost



For this example, imagine we split this node into two leaves.

Now let's calculate the **Similarity Scores** with **λ (lambda) = 1**.

$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

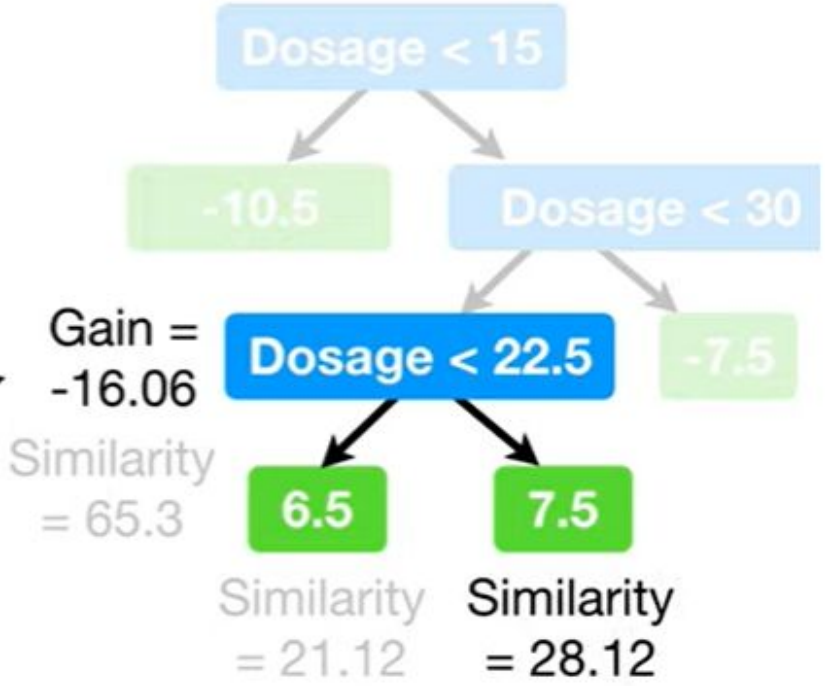
XGBoost

And for the right leaf, we get… **28.12**.

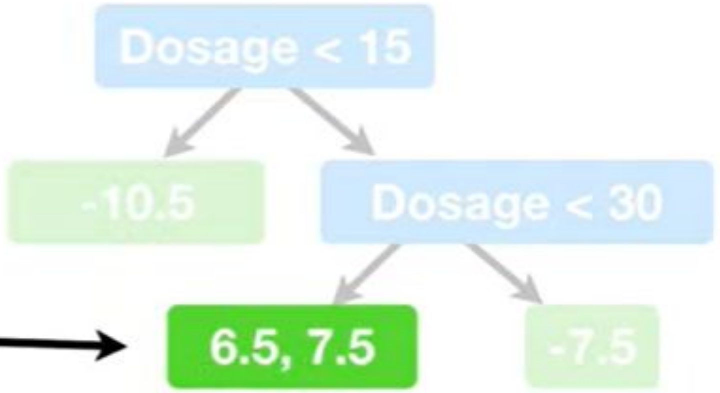
$$\text{Similarity Score} = \frac{7.5^2}{1 + 1} \boxed{= 28.12}$$

That means the **Gain** is… **-16.06**.

$$\text{Gain} = 21.12 + 28.12 - 65.3 = \boxed{-16.06}$$



XGBoost



…and we will prune this branch, even though **γ = 0**.

-16.07 - 0 = -16.07

Gain

γ (gamma)

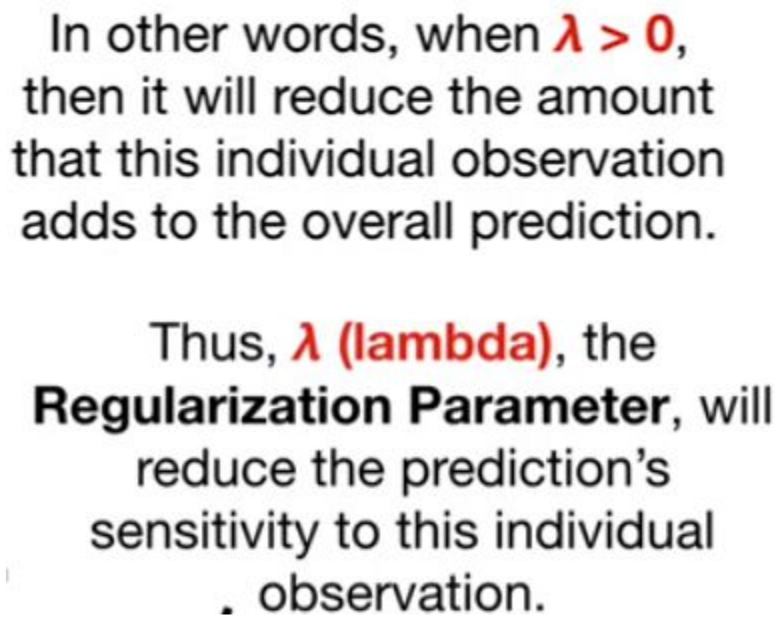
On the other hand, by setting **λ (lambda) = 1**, **λ** did what it was supposed to do; it prevented over fitting the **Training Data**.

XGBoost

In other words, setting $\gamma = 0$ does not turn off pruning.

On the other hand, by setting **λ (lambda) = 1**, **λ** did what it was supposed to do; it prevented over fitting the **Training Data**.

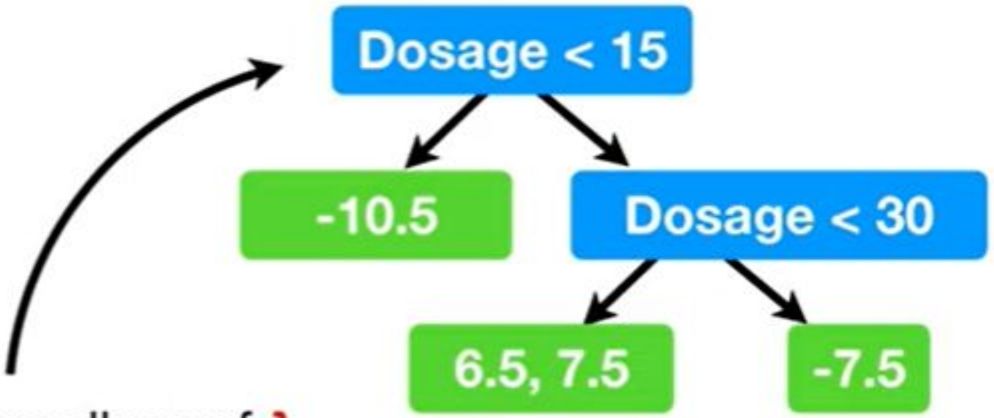
Gradient Boosting



In other words, when $\lambda > 0$, then it will reduce the amount that this individual observation adds to the overall prediction.

Thus, λ **(lambda)**, the **Regularization Parameter**, will reduce the prediction's sensitivity to this individual observation.

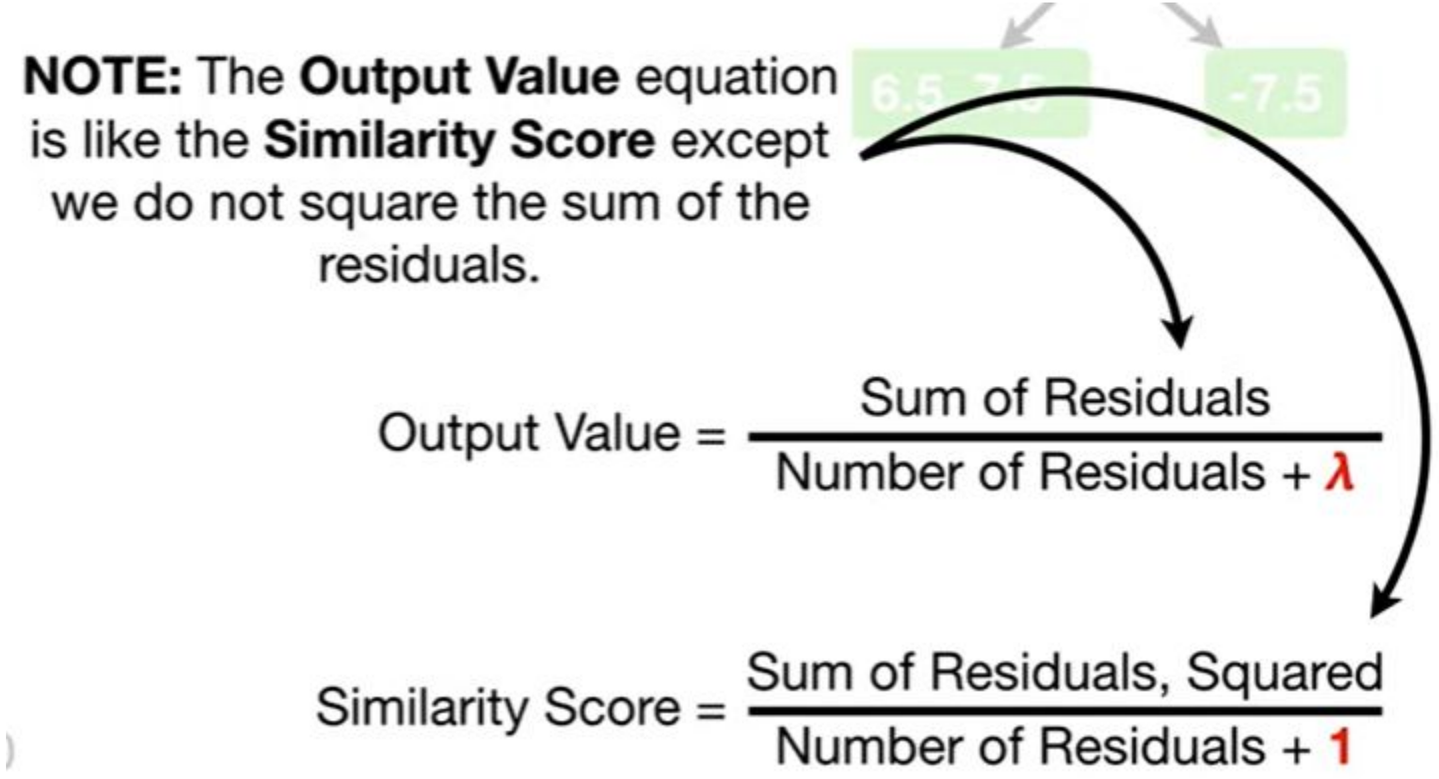
Gradient Boosting



For now, regardless of **λ (lambda)** and ***γ* (gamma)**, let's assume this is the tree we are working with…**this tree**

…and determine the **Output Values** for the leaves.

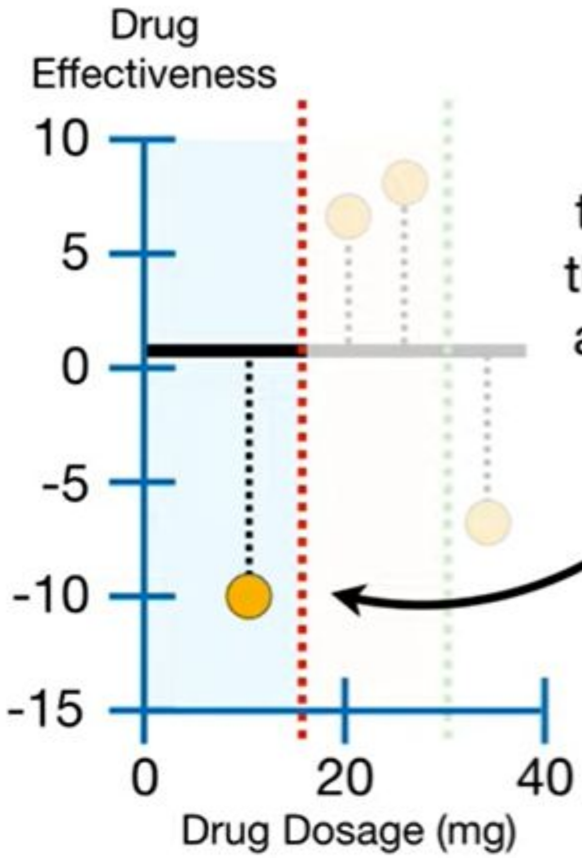
Gradient Boosting



NOTE: The **Output Value** equation is like the **Similarity Score** except we do not square the sum of the residuals.

$$\text{Output Value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

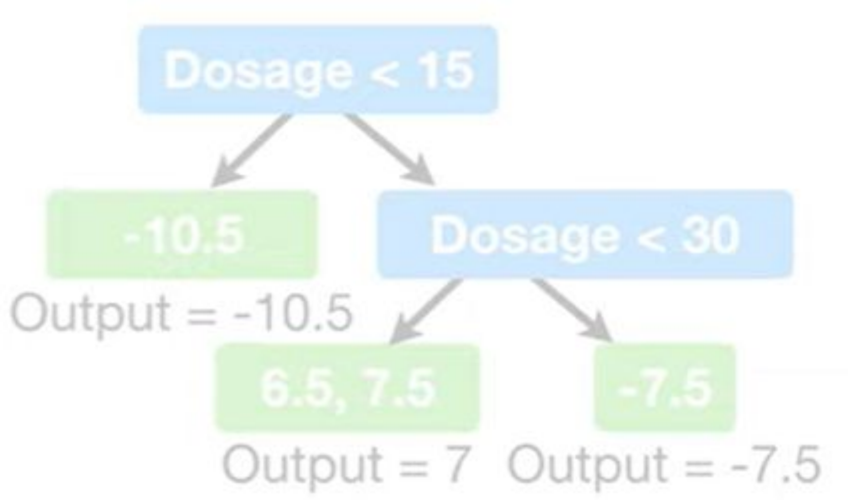
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

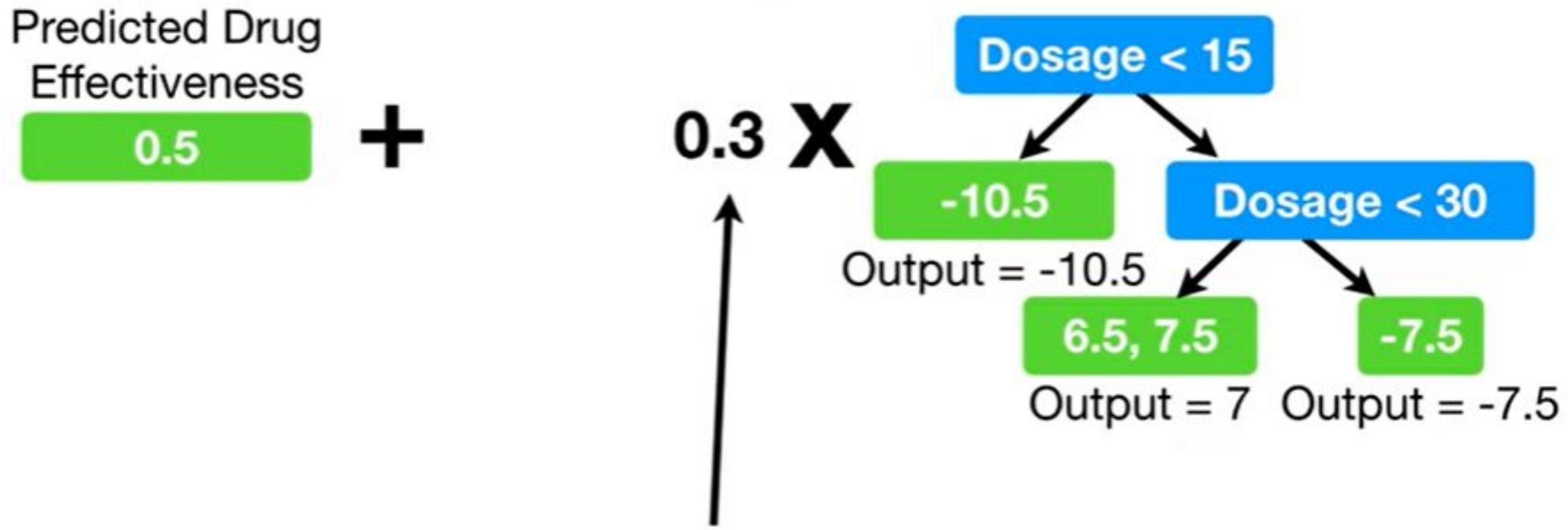


In other words, when **$\lambda > 0$**, then it will reduce the amount that this individual observation adds to the overall prediction.

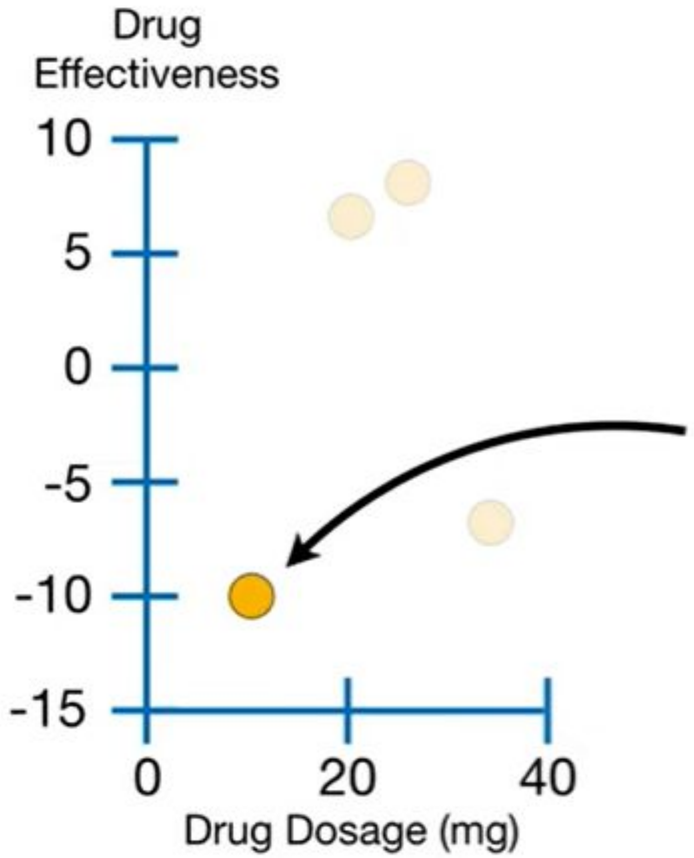
$$\text{Output Value} = \frac{-10.5}{1 + 1} = -5.25$$

Output values of leaf nodes with λ = 0





XGBoost calls the **Learning Rate**, ***ε*** **(eta)**, and the default value is **0.3**, so that's what we'll use.



Thus, the new **Predicted** value for this observation, with **Dosage = 10**…

Differences Between Bagging and Boosting –

S.NO	BAGGING	BOOSTING
1.	Simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by performance of previously built models.
5.	Different training data subsets are randomly drawn with replacement from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	Random forest.	Gradient boosting.