Università degli Studi di Padova

Machine Learning
2024/2025

AMCO
Artificial Intelligence, Machine Learning and Control Research Group

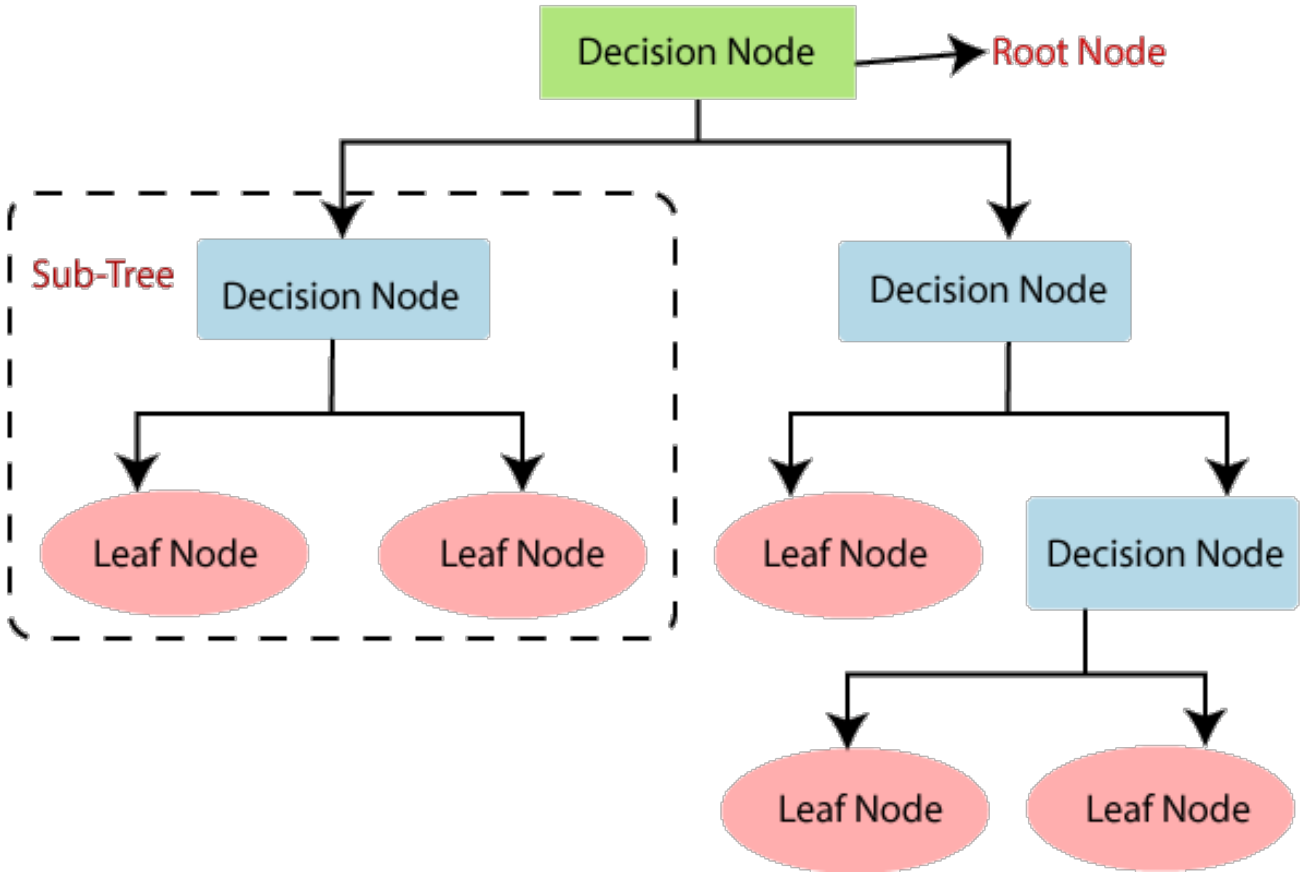
Lecture #17 Ensemble Tree-based Approaches

Gian Antonio Susto

Recap: the Decision Tree

At the heart is the decision tree, a structure that mimics human decision-making by splitting data into branches based on feature values.

Each internal node of the tree represents a decision based on a feature, each branch represents the outcome of that decision, and each leaf node corresponds to a prediction or outcome.

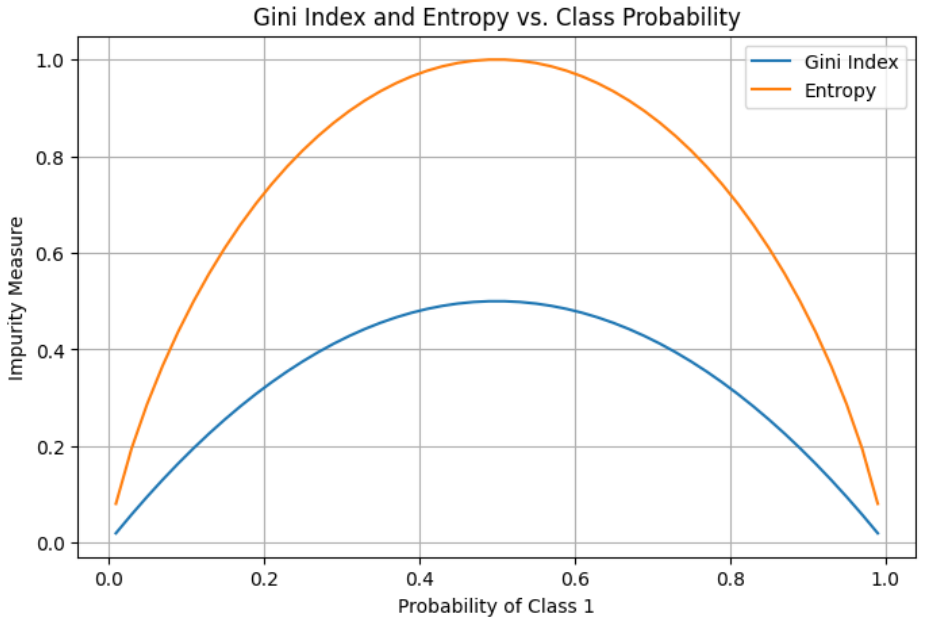


Recap: Gini Index / Entropy / Information Gain

The Gini Index (or Gini Impurity) is a measure of how impure or mixed a dataset is.

For a dataset S with c classes:

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

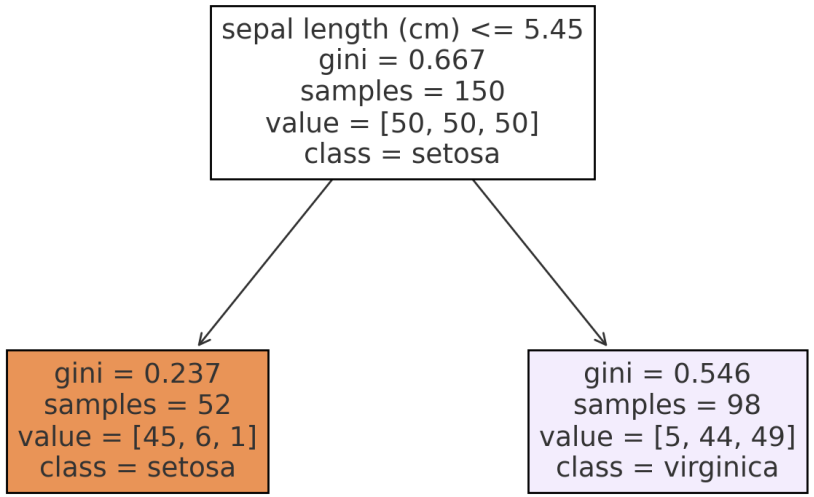


$$Gini_{split} = \frac{n_{left}}{n} \cdot Gini(left) + \frac{n_{right}}{n} \cdot Gini(right)$$

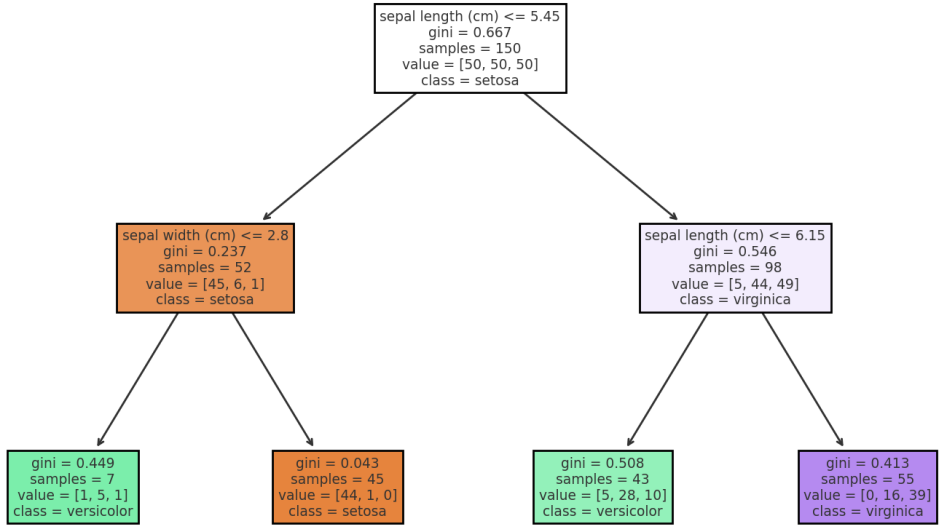
$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

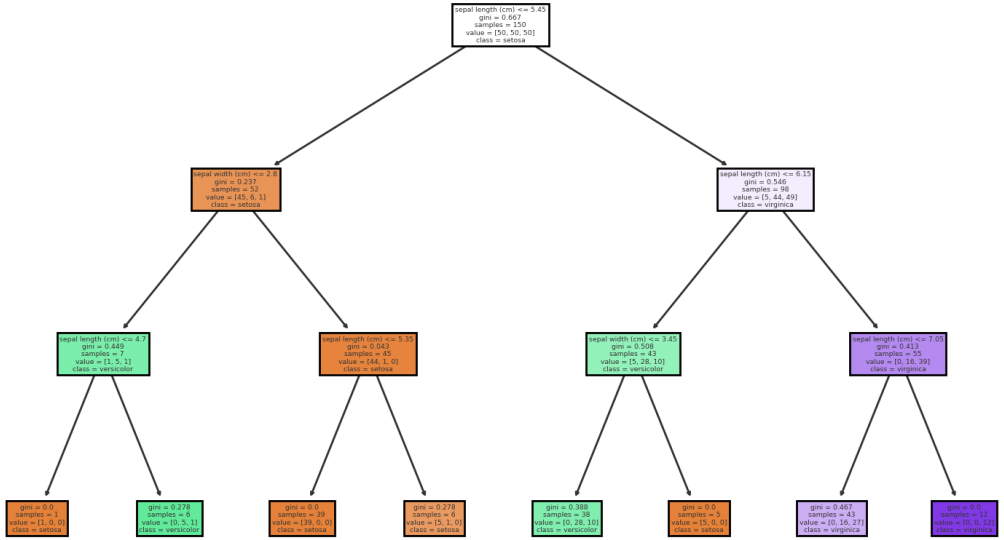
max_depth = 1



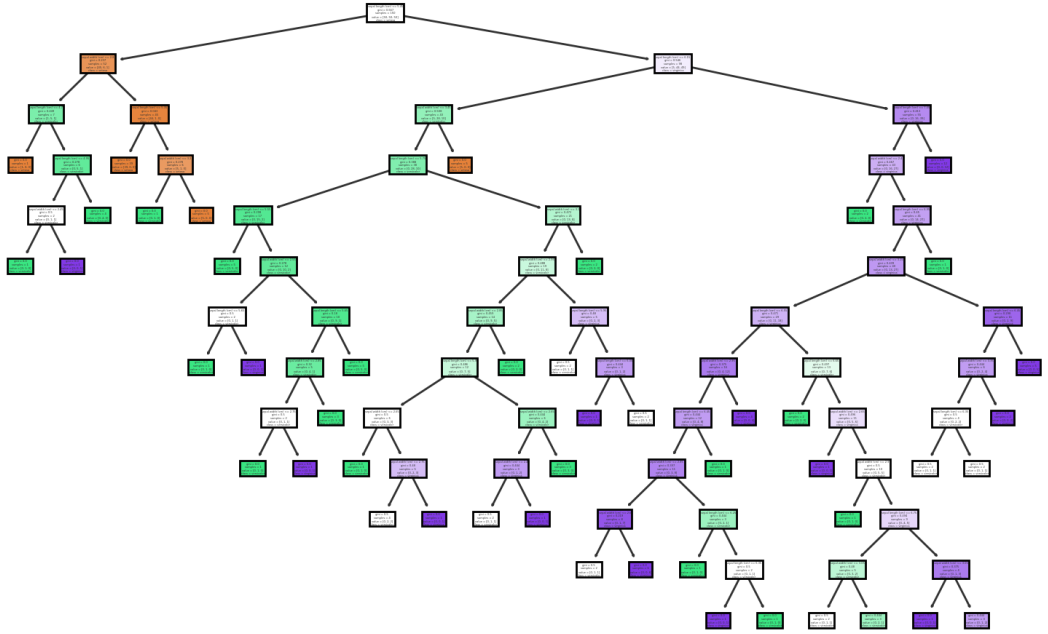
max_depth = 2

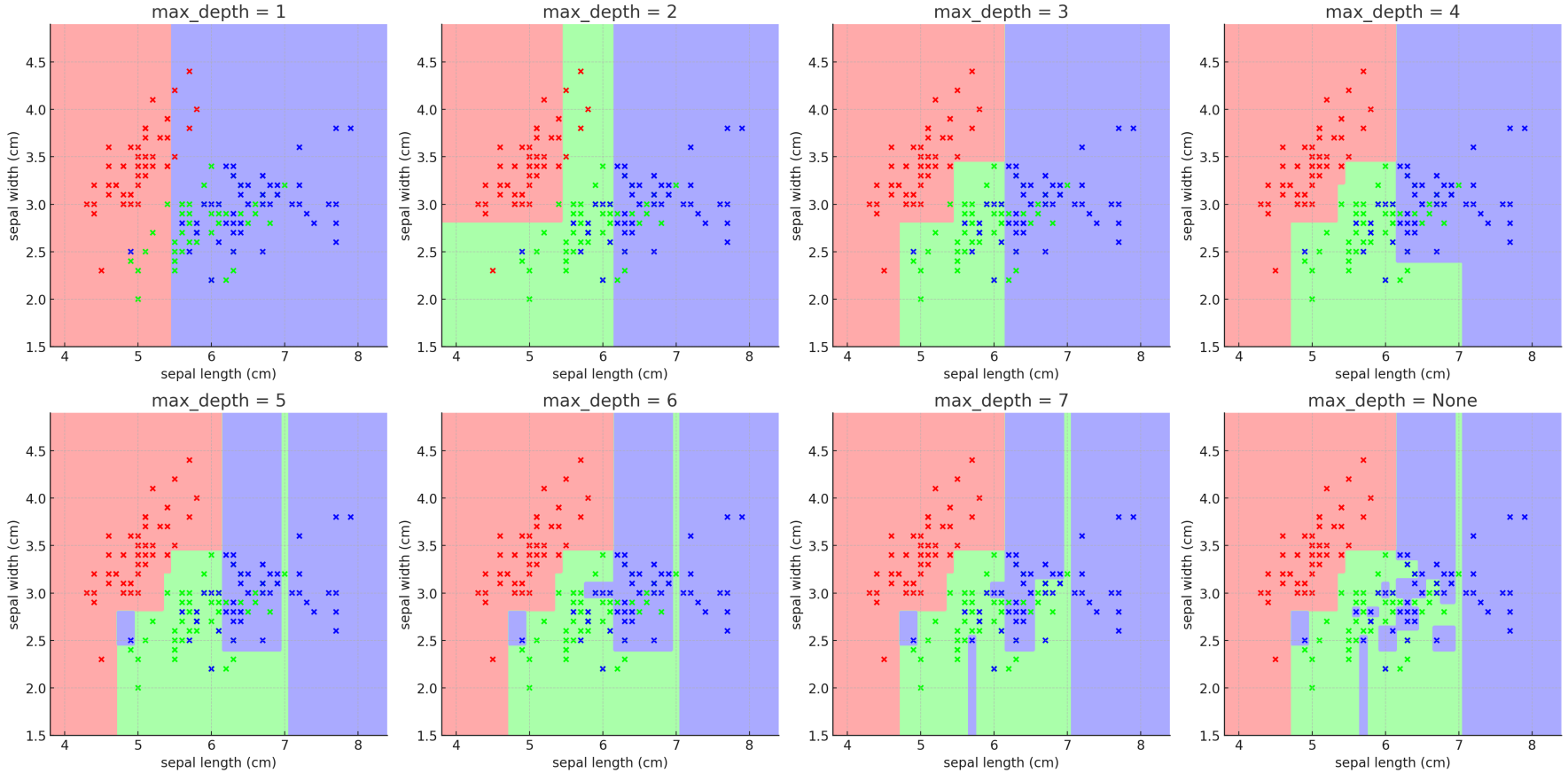


max_depth = 3

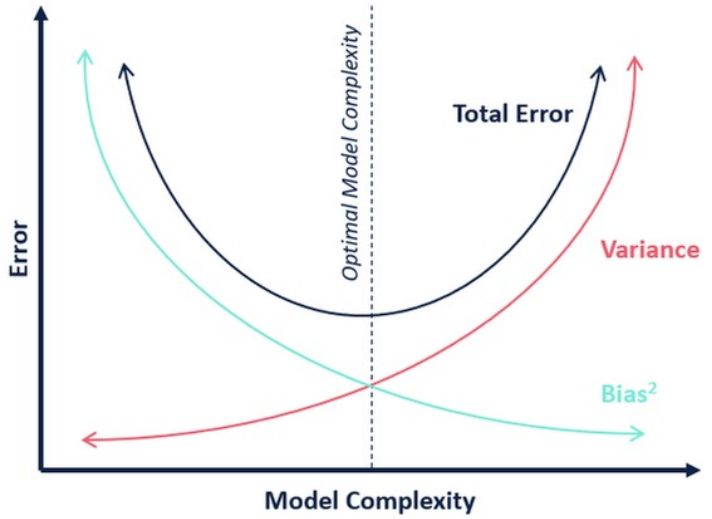


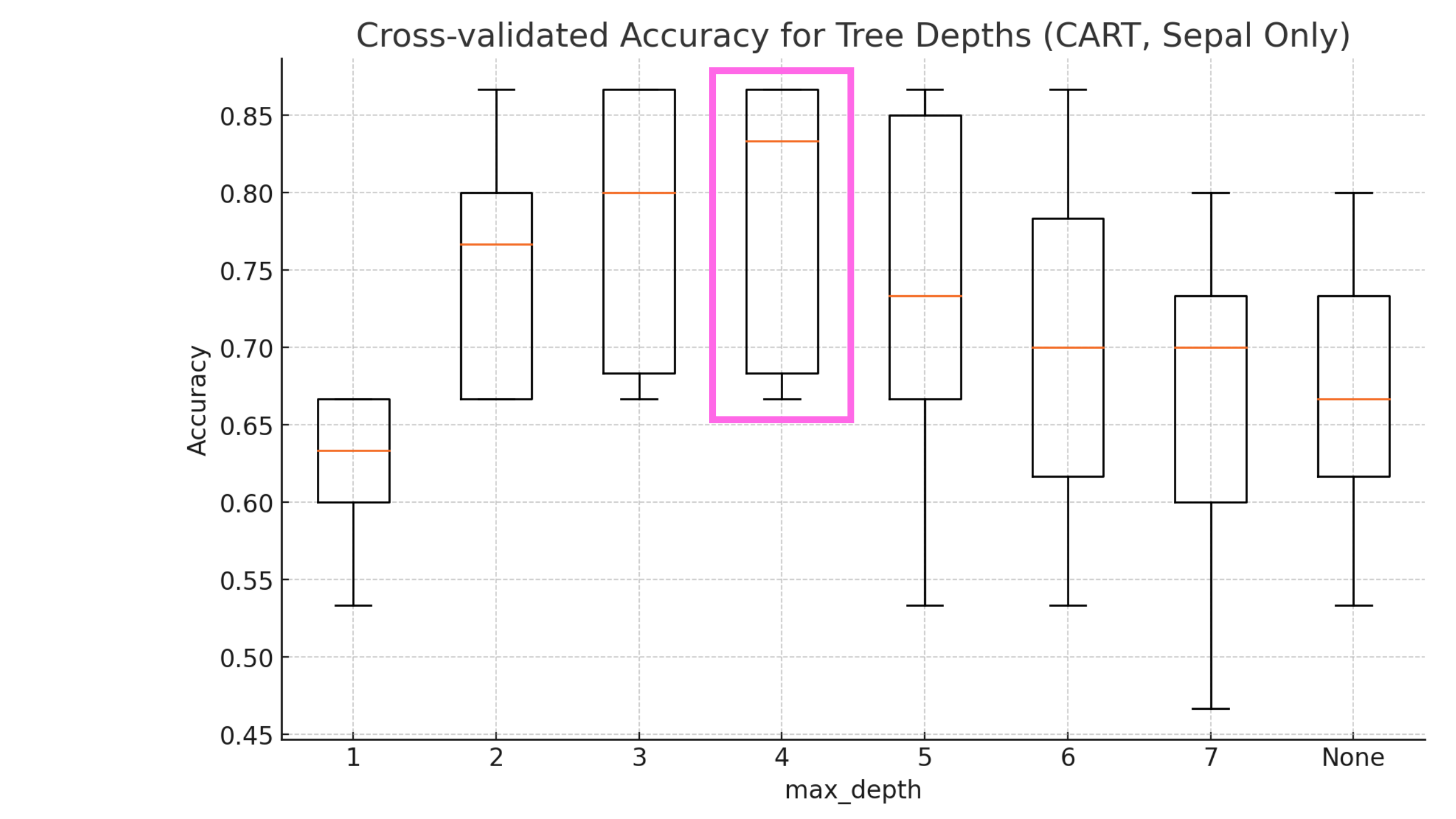
max_depth = None (full tree)





Don’t we see a similar behaviour that we had with other algorithms?




Cross-validated Accuracy for Tree Depths (CART, Sepal Only)
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
1
2
3
4
5
6
7
None
max_depth

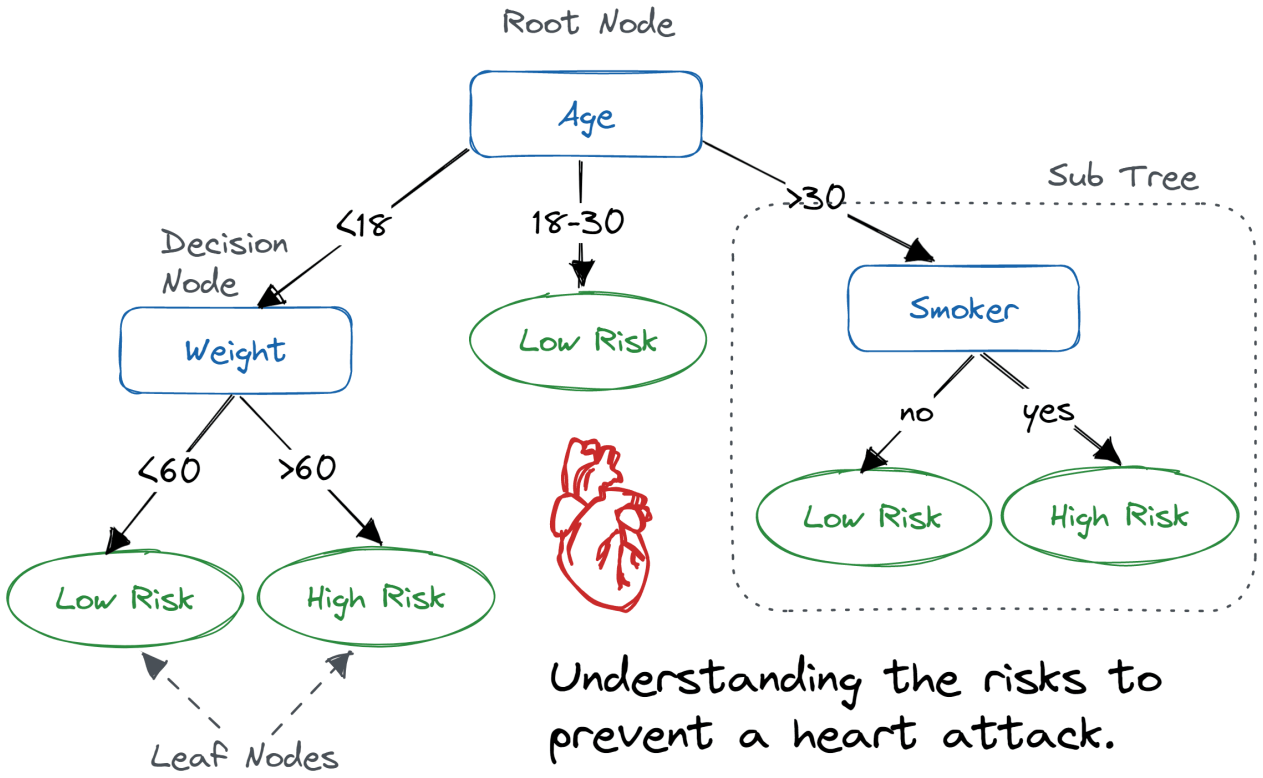
Recap: Decision Tree

Advantages

- Easily interpretable
- They require no data normalization
- The classification is almost immediate
- The computational expensive part is done off-line (once)

Drawbacks

- Really high variance classifiers→ Prone to overfitting! Typically, poor generalization performances!



🌳 Recap: tree-based Approaches

Tree-based methods are among the most effective techniques for supervised learning, particularly when working with smaller datasets (with *n* fewer than 10,000 samples).

Interestingly, the core concepts behind them are quite straightforward...

Why do tree-based models still outperform deep learning on typical tabular data? Part of Advances in Neural Information Processing Systems 35 (NeurIPS 2022)

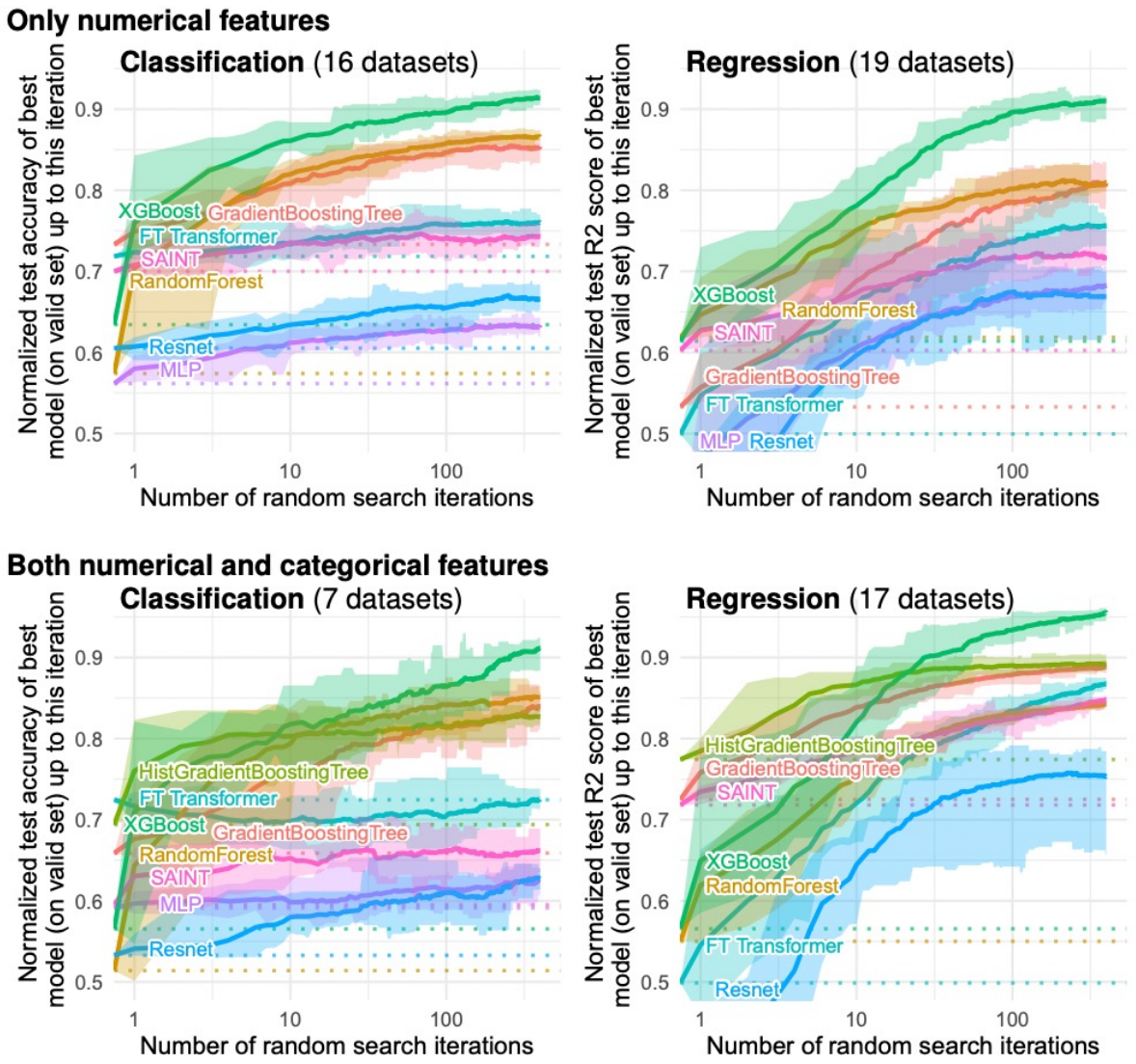


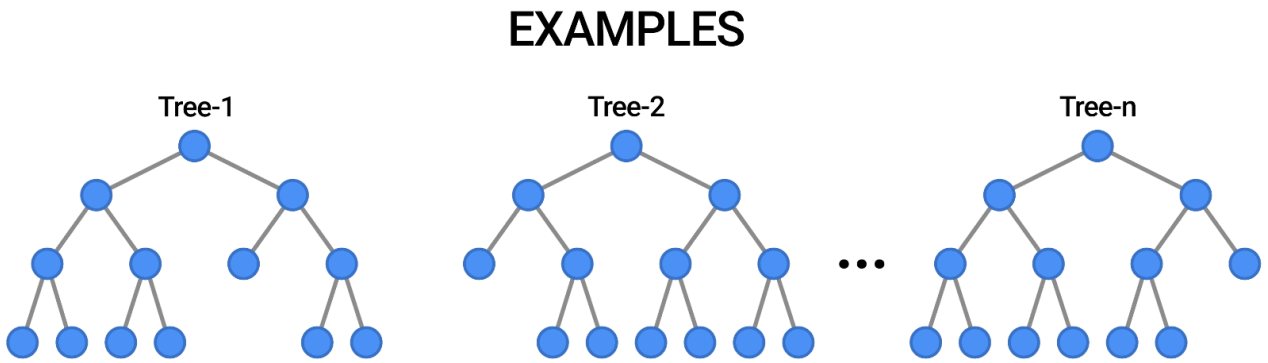
Figure 1: **Benchmark on medium-sized datasets**, top only numerical features; bottom: all features. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to minimum and maximum scores on these 15 shuffles.

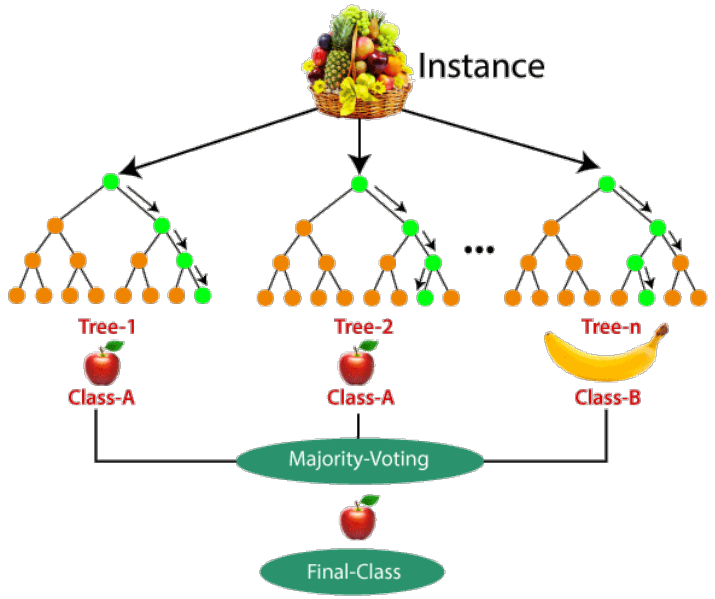
Recap: Random Forest (RF)

A RF is composed by many ‘weak’ learners (decision trees): we cleverly combine DTs reducing overfitting!

We construct slightly different DTs (more on this later) and, in classification, we decide by a majority-voting (we choose following the mode) the final class. In regression, the final decision is the average.

This in an ‘ensemble’ approach: we combine multiple models (often called base learners or weak learners) to produce a stronger model.



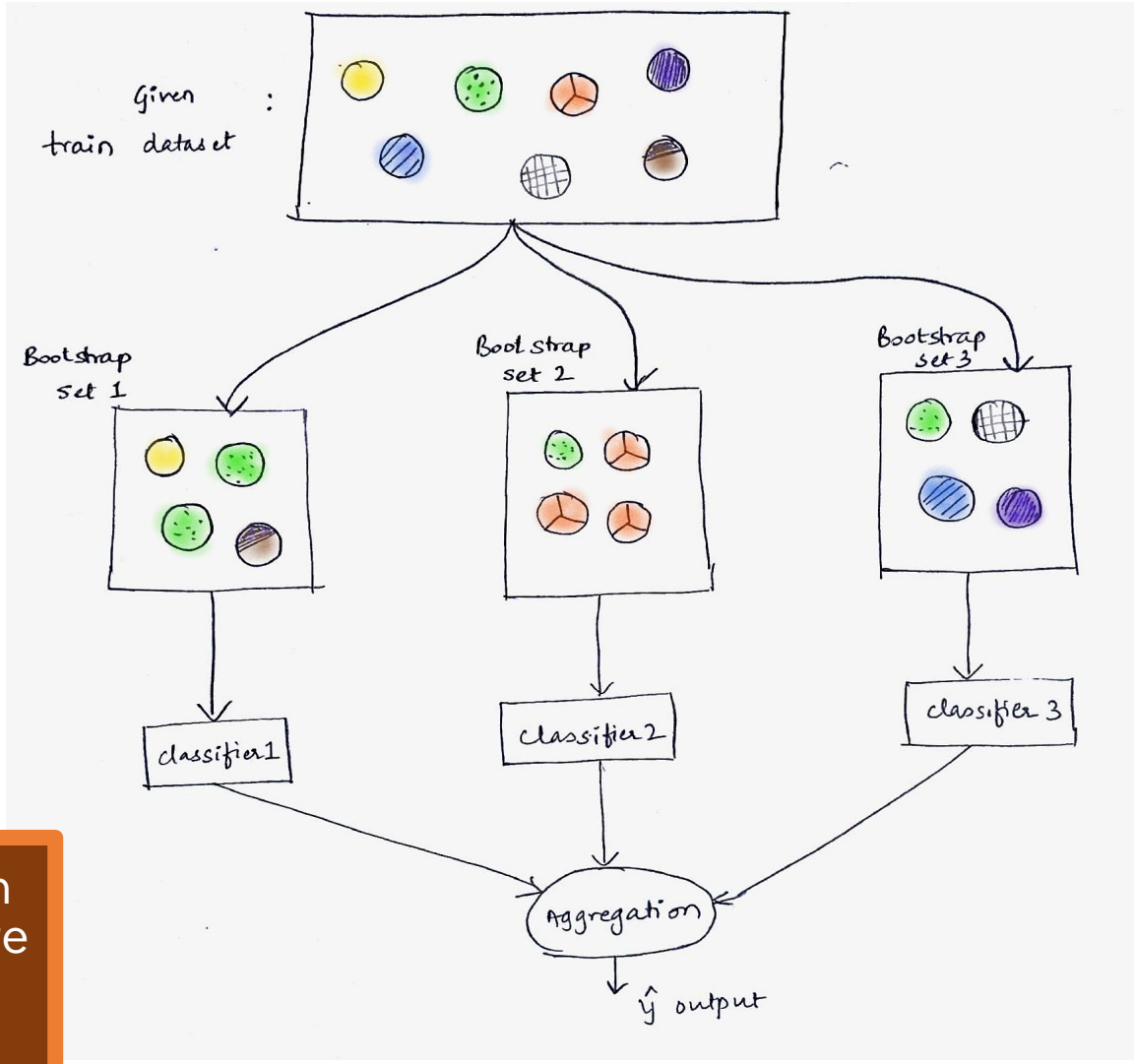


🔧 Recap: How to Build a Random Forest

Let's assume you want to build a forest with T trees.

For each tree:

- Sample the dataset with replacement (bootstrap sample). This procedure is called Bagging (bootstrap aggregating).
- Build a decision tree: but at each split, instead of evaluating all features, pick a random subset (e.g., $\sqrt{p}$). This procedure is called Feature Bagging.



The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the T trees, causing them to become correlated.

RF: feature importance

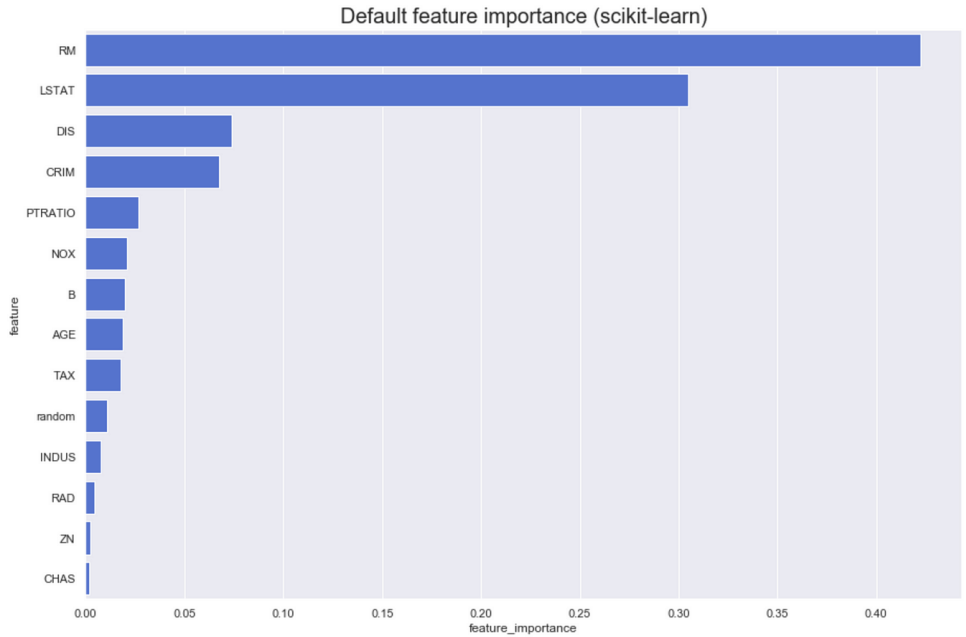
Feature importance reflects how useful or valuable each feature is for making predictions in a model. For decision trees (and ensembles like Random Forests), it's typically based on:

🔍 How much each feature decreases impurity (e.g., Gini index or entropy) when it’s used to split the data

📊 Intuition

- If a feature is consistently chosen for important splits (i.e., it helps reduce impurity a lot), it gets high importance.

- Features that are rarely used or don’t reduce impurity much get low or zero importance.



RF: feature importance

Feature importance reflects how useful or valuable each feature is for making predictions in a model. For decision trees (and ensembles like Random Forests), it's typically based on:

🔍 How much each feature decreases impurity (e.g., Gini index or entropy) when it’s used to split the data

📊 Intuition

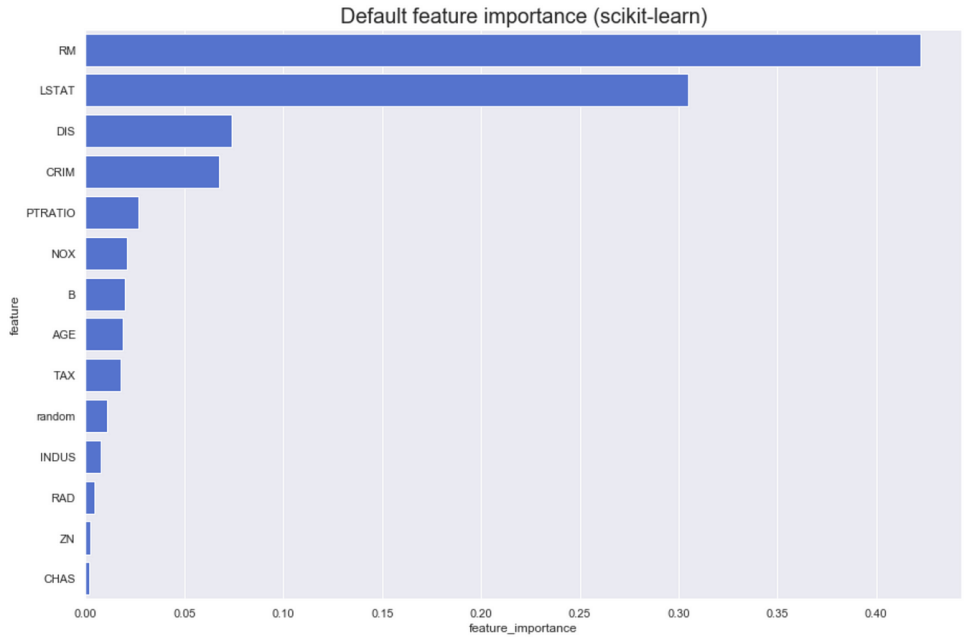
- If a feature is consistently chosen for important splits (i.e., it helps reduce impurity a lot), it gets high importance.

- Features that are rarely used or don’t reduce impurity much get low or zero importance.

This is a ‘eXplainable Artificial Intelligence (XAI)’ approach.

It is a ’global’ approach: provide us with info on the whole model structure

Any idea how can this information be exploited?



RF: feature importance - Derivation

Let's consider the Gini impurity, and we have a decision tree:

1. At every split, the algorithm calculates how much that split reduces impurity:

$$\Delta Gini = Gini(\text{parent}) - \left(\frac{n_{left}}{n_P} \cdot Gini(\text{left}) + \frac{n_{right}}{n_P} \cdot Gini(\text{right})\right)$$

2. The contribution of a feature is the sum of all impurity decreases where that feature was used to split:

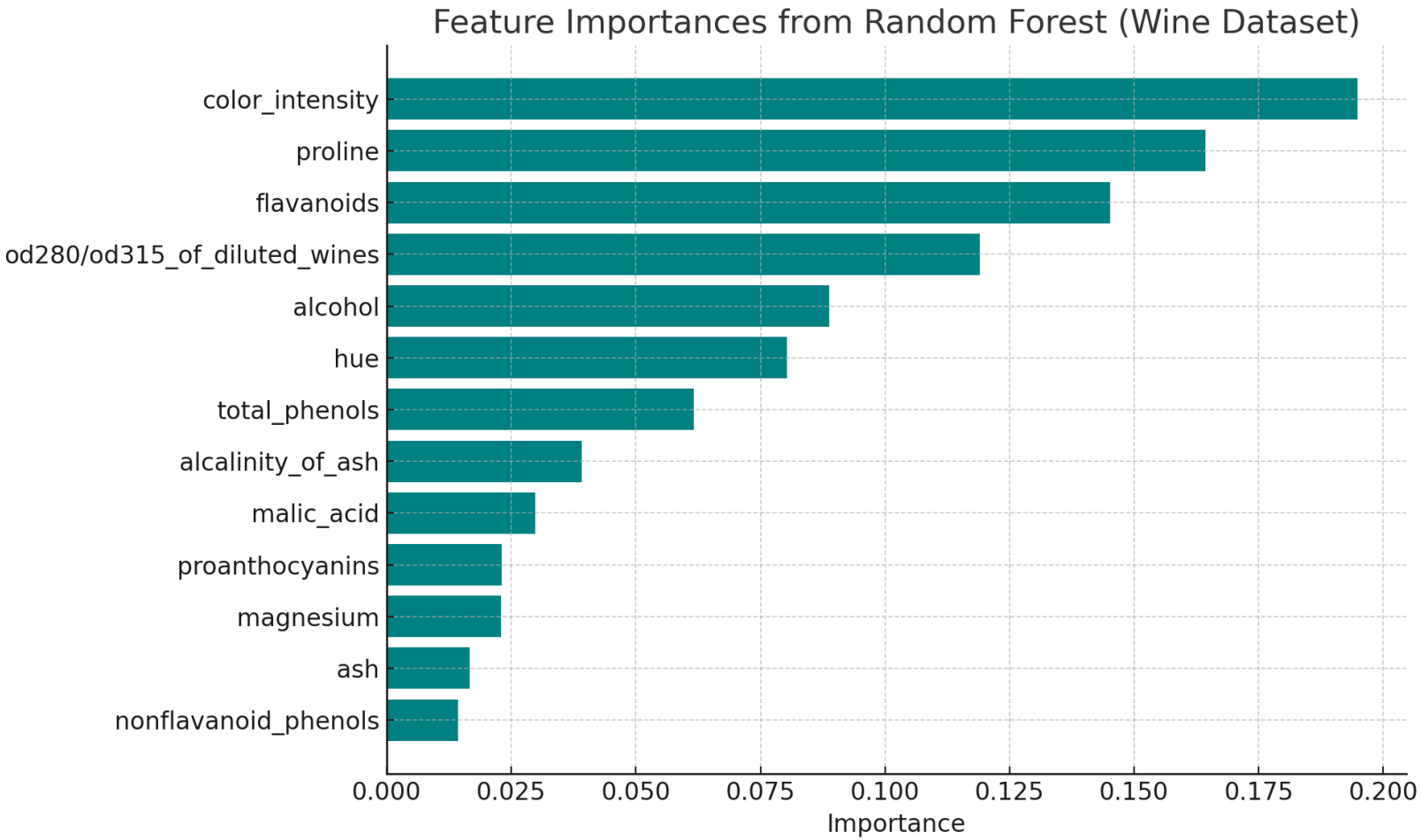
$$Importance(feature) = \sum_{\text{nodes using feature}} \frac{n_p}{n_{TOT}} \Delta Gini$$

3. In a Random Forest, we average this importance over all the trees in the forest.
4. (Optional) the imp

$$\text{Normalized Importance} = \frac{\text{Raw Importance}}{\sum \text{Raw Importances}}$$

Where:
n_p is the number of samples at the parent node
n_{TOT} is the total number of samples

On the wine dataset



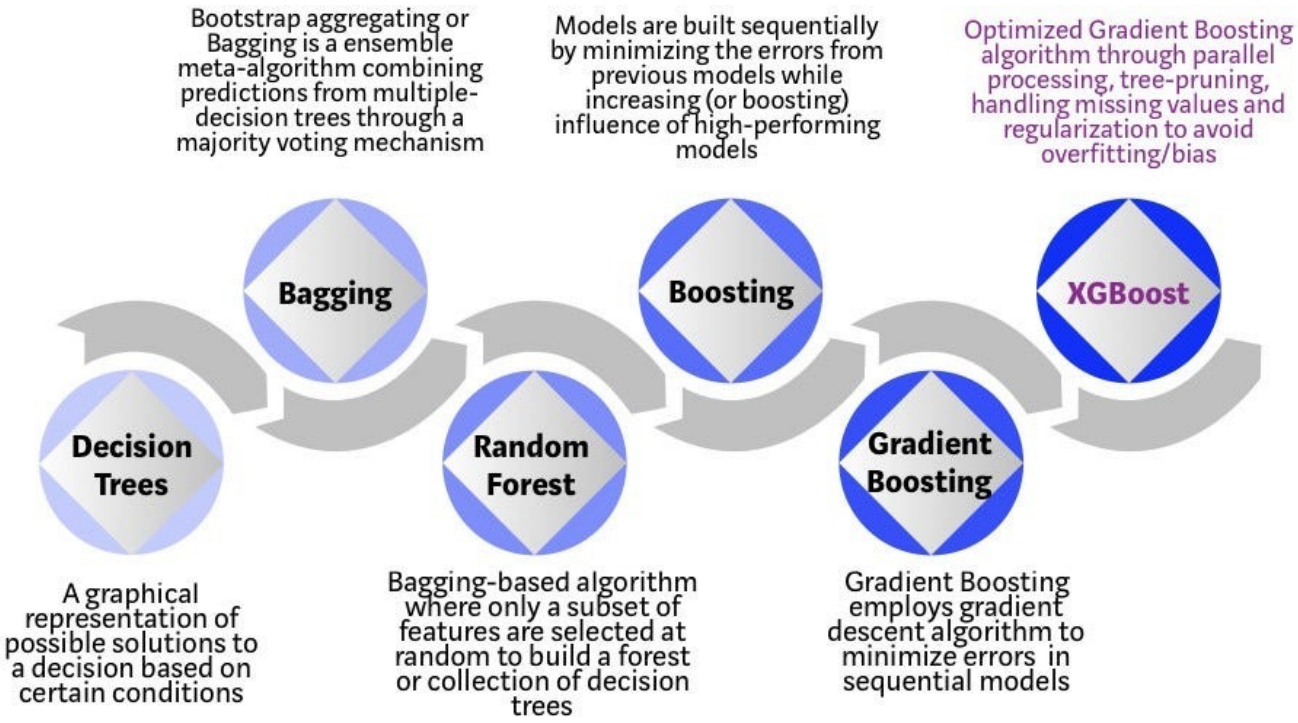
RF is not the only tree-based ensemble approach!

Many approaches in the literature! All these methods aim to overcome the overfitting problem of individual decision trees by:

✅ Building multiple trees

✅ Imposing constraints (on data, features, model structure, or learning process)

✅ Combining their outputs to produce more robust and generalizable predictions



RF is not the only tree-based ensemble approach!

Many approaches in the literature! All these methods aim to overcome the overfitting problem of individual decision trees by:

✅ Building multiple trees

✅ Imposing constraints (on data, features, model structure, or learning process)

✅ Combining their outputs to produce more robust and generalizable predictions

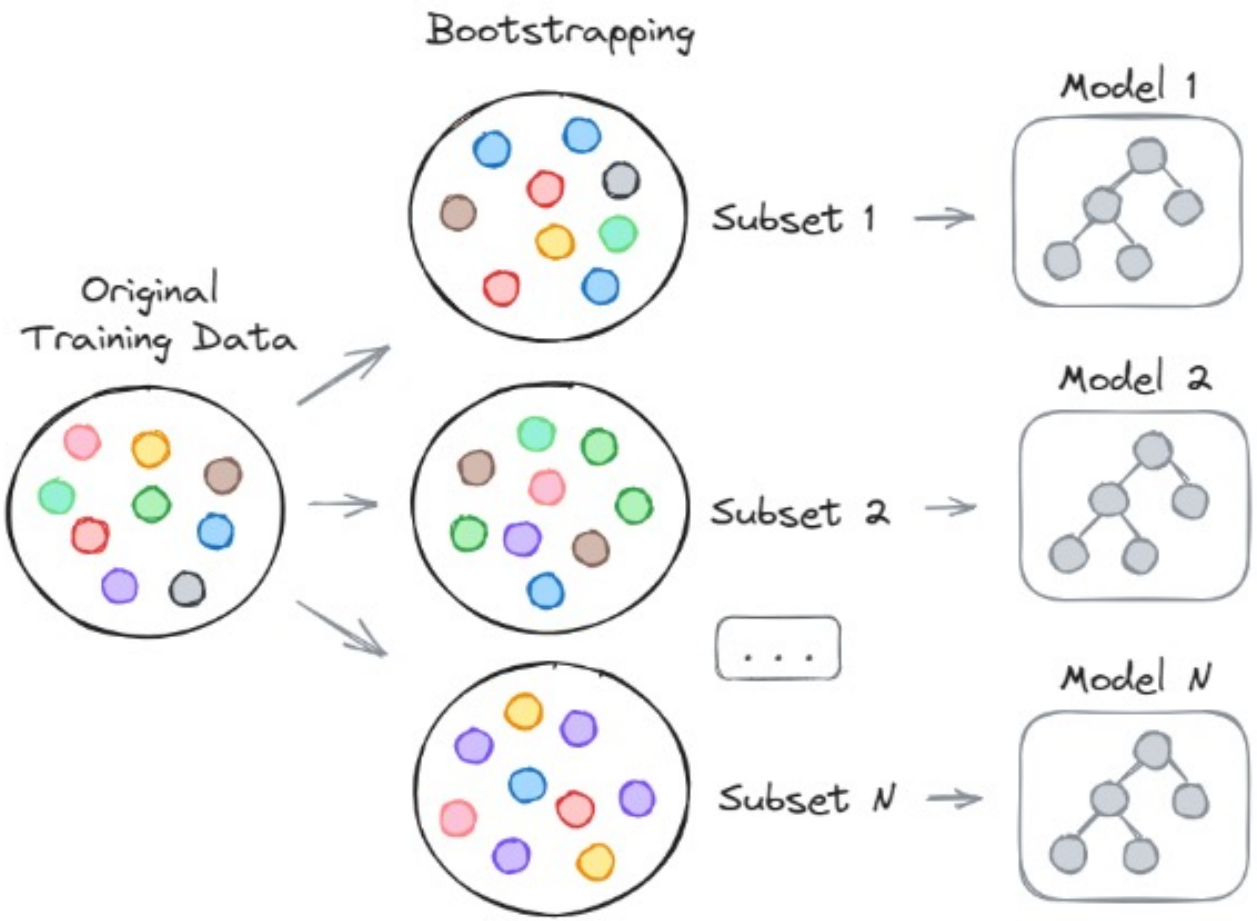
What varies between methods:

Dimension	Examples of Constraints or Strategies
How trees are built	Independently (Bagging), Sequentially (Boosting)
Data sampling	Random subsets of rows (Bagging), Weighted sampling (AdaBoost), Full data with residuals (GBM)
Feature usage	All features (Bagging), Random subset per split (Random Forest)
Tree complexity	Deep trees (Random Forest), Shallow trees/stumps (AdaBoost), Controlled depth (GBM variants)
Loss optimization	Classification error (AdaBoost), Gradient of loss (GBM, XGBoost), Log-loss or custom losses
Regularization	No regularization (RF), Explicit penalties (XGBoost), Leaf-wise constraints (LightGBM)

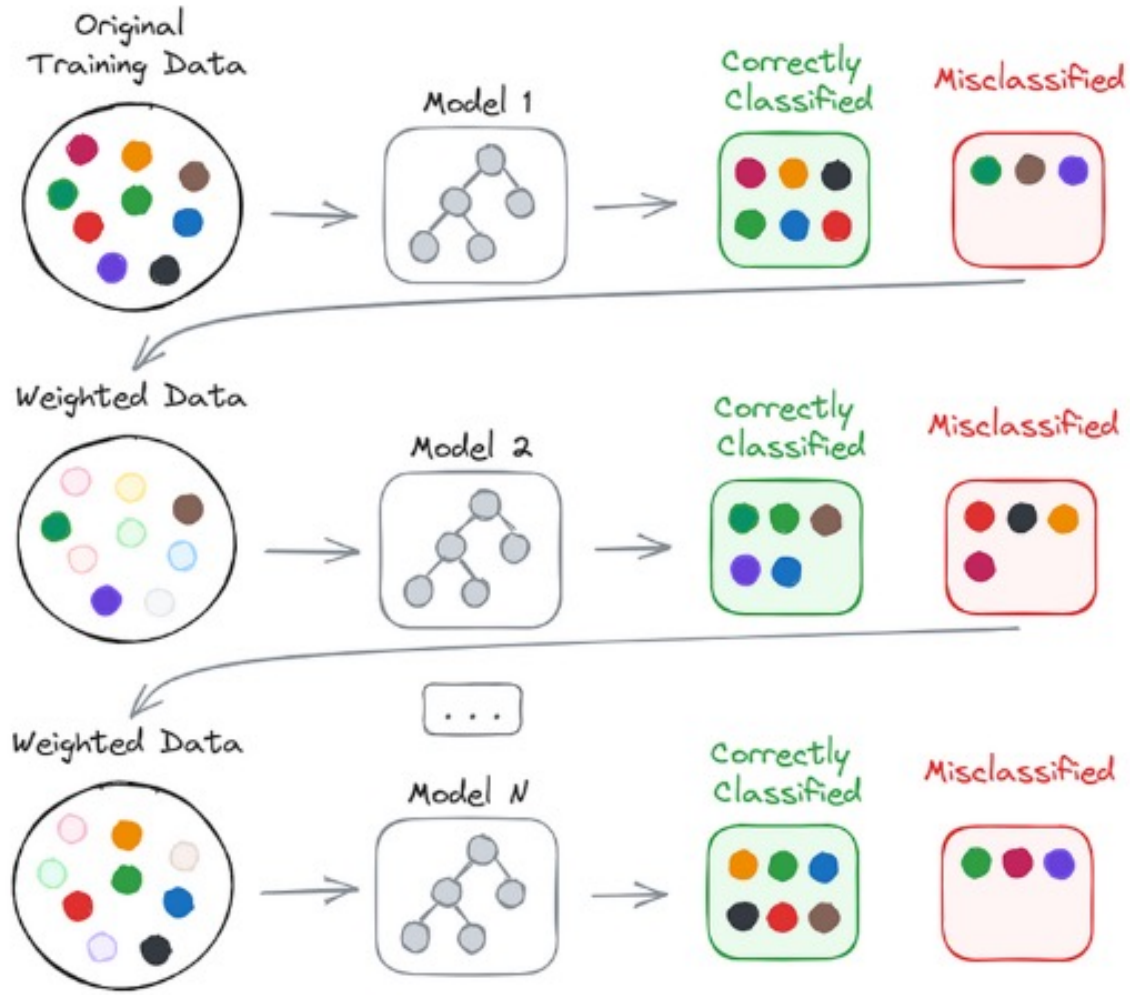
Method	Core Idea	Model Combination	Key Traits
Bagging	Train trees on random data subsets	Averaging / Voting	Reduces variance, parallelizable, robust to overfitting
Random Forest	Bagging + random feature selection	Averaging / Voting	Strong baseline, good generalization
Boosting	Sequential models to fix previous errors	Weighted sum	Reduces bias, sensitive to noise
AdaBoost	Focus on misclassified samples via reweighting	Weighted sum	Simple, uses weak learners (e.g., stumps), effective on clean data
Gradient Boosting	Fit to loss function gradients	Weighted sum	Flexible loss functions, can overfit without tuning
XGBoost	Regularized GBM with pruning and optimizations	Weighted sum	Fast, regularized, handles missing values
LightGBM	Histogram-based GBM, leaf-wise growth	Weighted sum	Very fast, memory-efficient, great for large-scale problems
CatBoost	Categorical-feature-friendly GBM	Weighted sum	Handles categoricals natively, avoids overfitting
Stacked Ensemble	Combine diverse models with meta-learner	Meta-model (e.g., regression)	Very flexible, risk of overfitting without proper cross-validation

Method	Core Idea	Model Combination	Key Traits
Bagging	Train trees on random data subsets	Averaging / Voting	Reduces variance, parallelizable, robust to overfitting
Random Forest	Bagging + random feature selection	Averaging / Voting	Strong baseline, good generalization
Boosting	Sequential models to fix previous errors	Weighted sum	Reduces bias, sensitive to noise
AdaBoost	Focus on misclassified samples via reweighting	Weighted sum	Simple, uses weak learners (e.g., stumps), effective on clean data
Gradient Boosting	Fit to loss function gradients	Weighted sum	Flexible loss functions, can overfit without tuning
XGBoost	Regularized GBM with pruning and optimizations	Weighted sum	Fast, regularized, handles missing values
LightGBM	Histogram-based GBM, leaf-wise growth	Weighted sum	Very fast, memory-efficient, great for large-scale problems
CatBoost	Categorical-feature-friendly GBM	Weighted sum	Handles categoricals natively, avoids overfitting
Stacked Ensemble	Combine diverse models with meta-learner	Meta-model (e.g., regression)	Very flexible, risk of overfitting without proper cross-validation

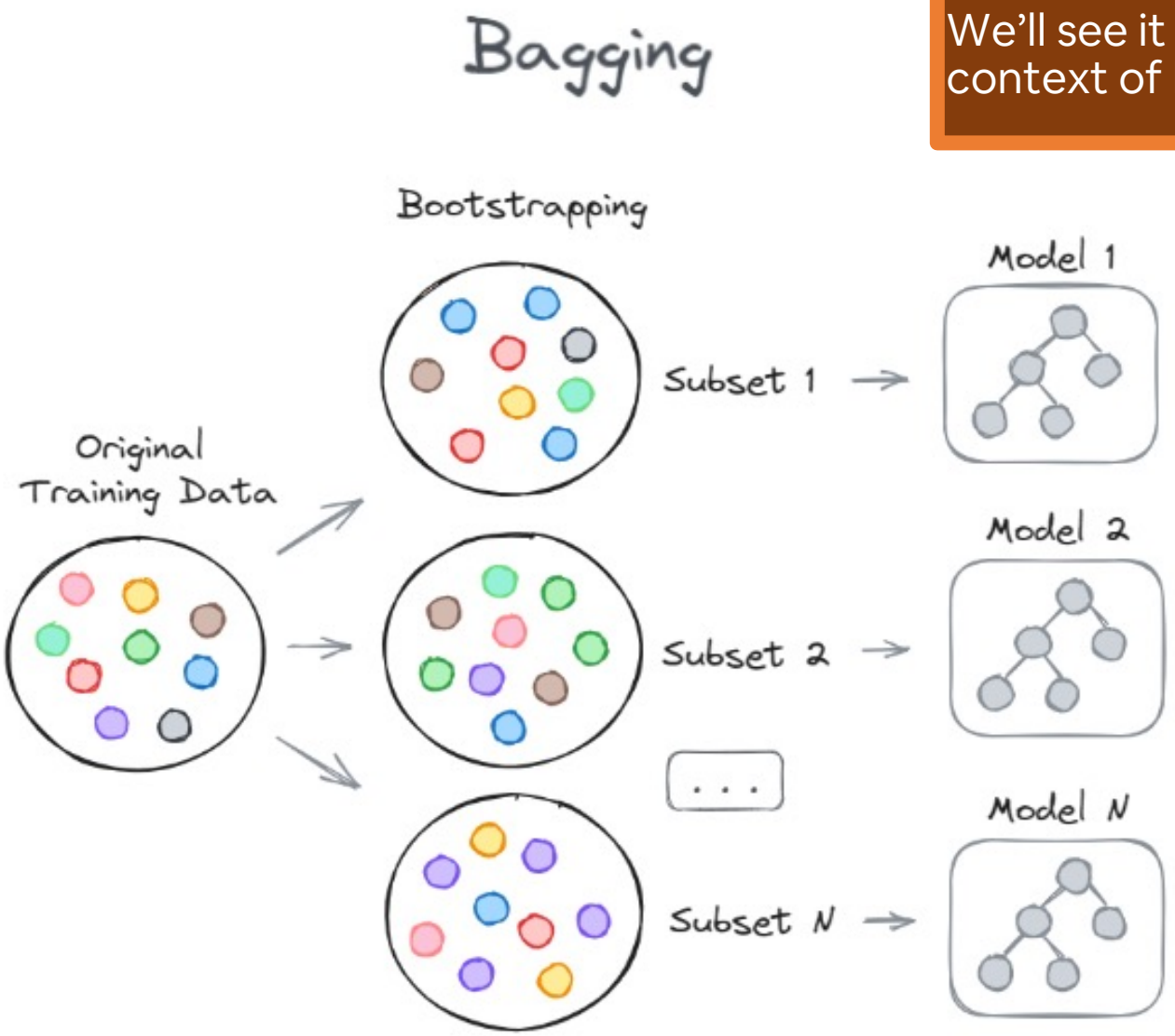
Bagging

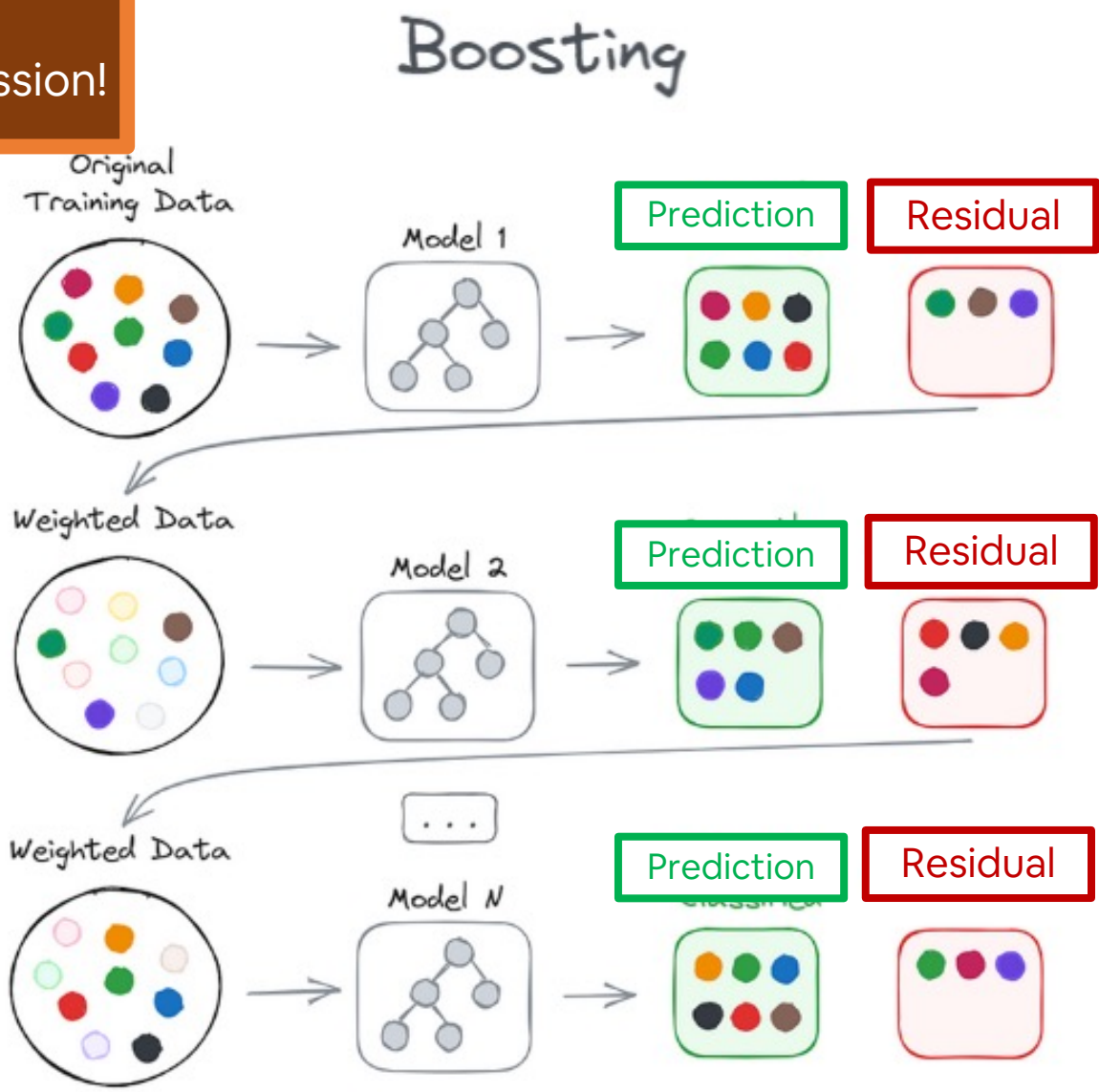


Boosting



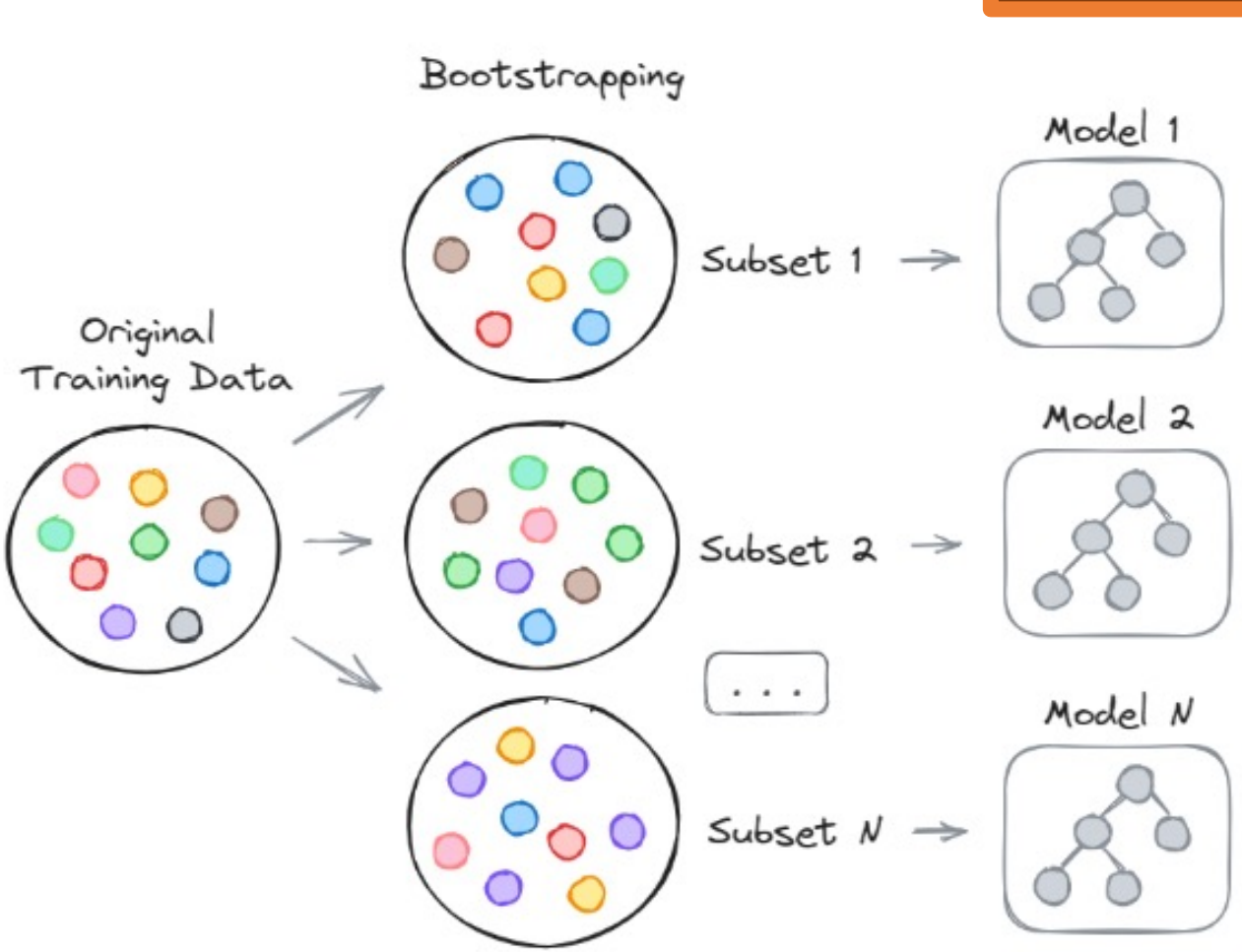
We'll see it in the context of Regression!



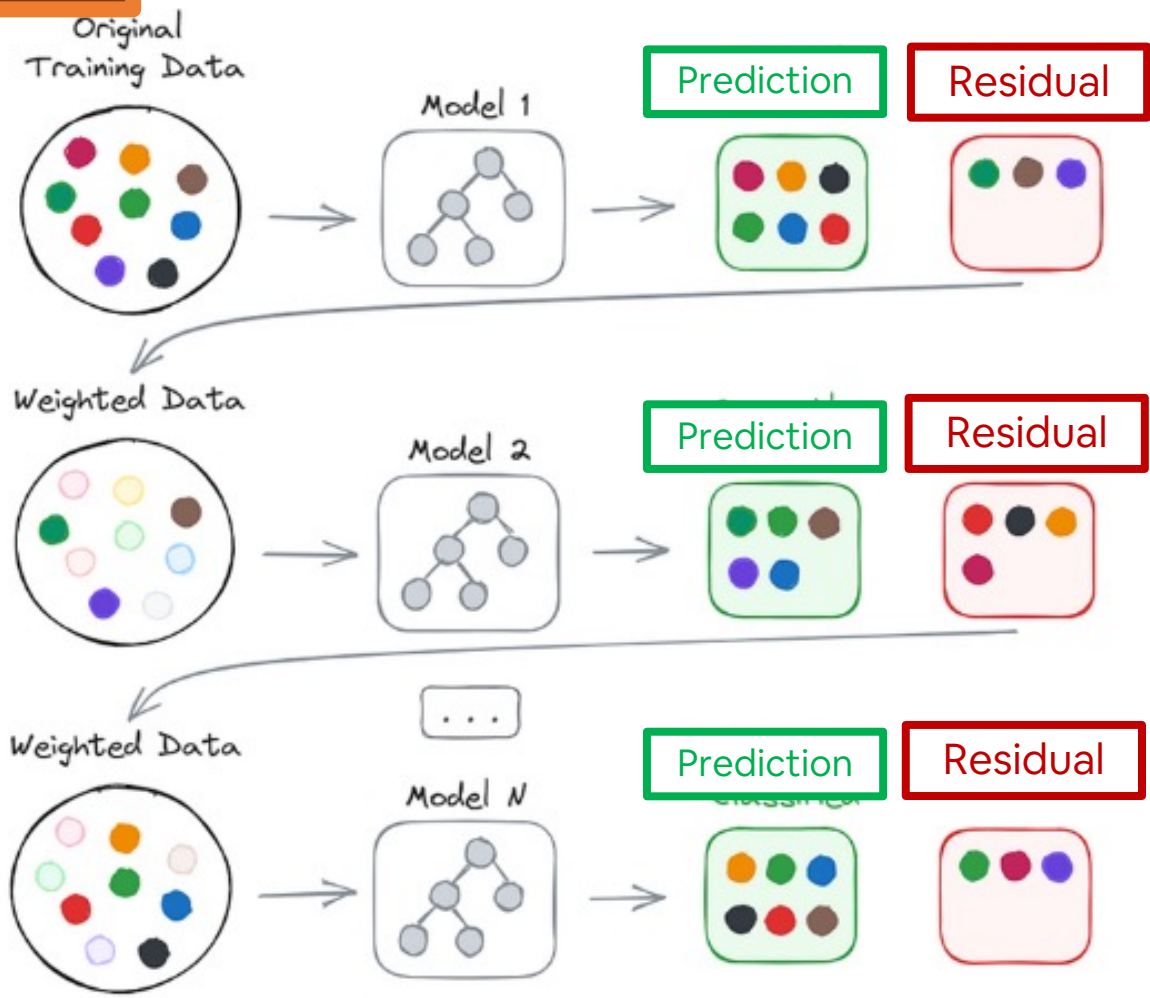


Bagging

We'll see it in the context of Regression!



Boosting



Does this approach remind you of something we have seen in the past?

Why is it called gradient boosting?

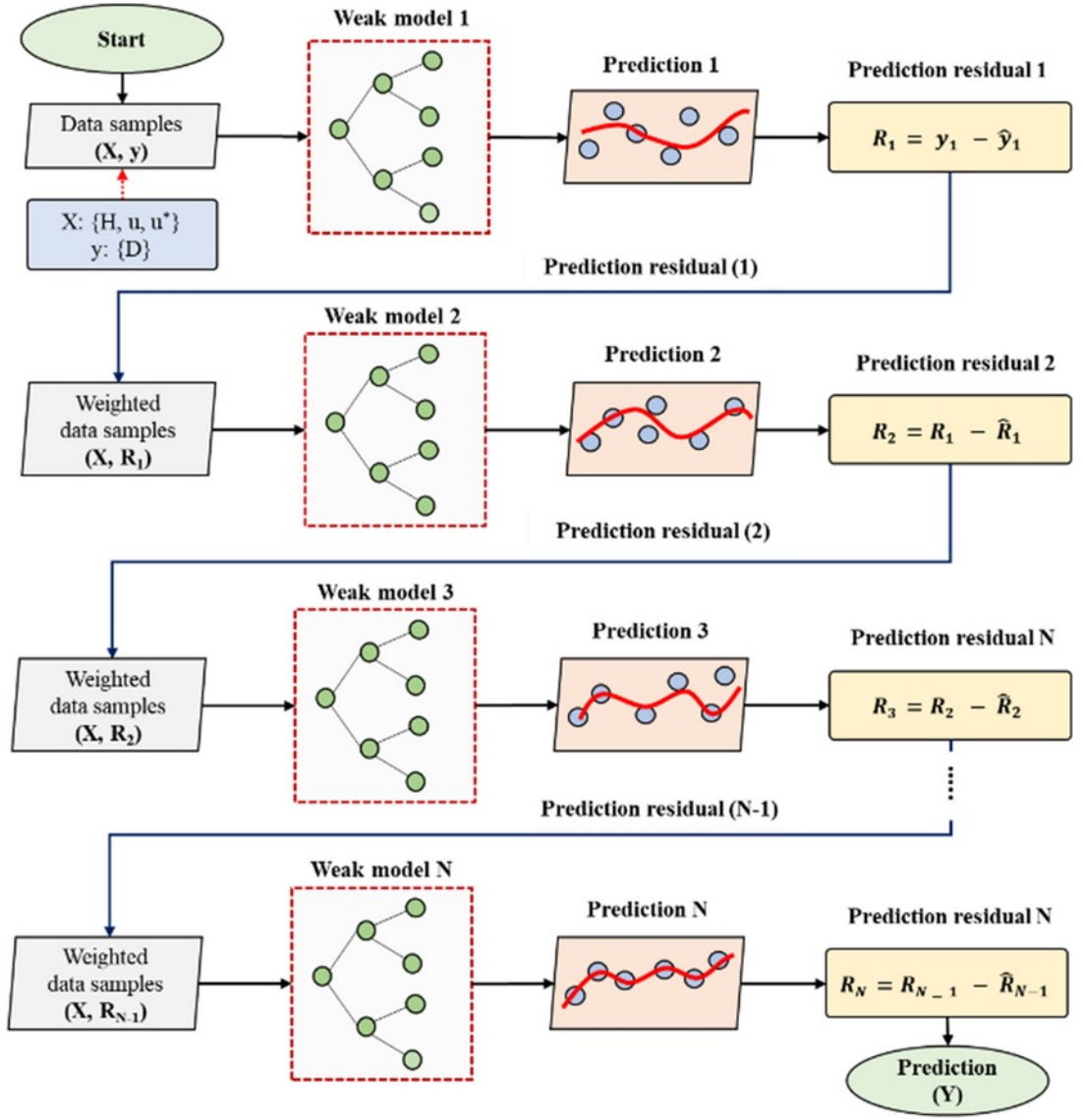
🔷 Boosting

Boosting is a method that builds a **strong learner** (accurate model) by combining many **weak learners** (typically decision trees with few splits). The idea is to train models sequentially, each one trying to **correct the errors** of the previous ones.

🔷 Gradient

'Gradient' refers to the use of gradient descent, a mathematical optimization technique. In the context of gradient boosting, it's used to minimize a loss function (like MSE or log loss) by fitting new models to the negative gradients of the loss function — which are essentially the residual errors.

In other words, each new tree is trained to predict the gradient of the loss function with respect to the current model's predictions



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
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

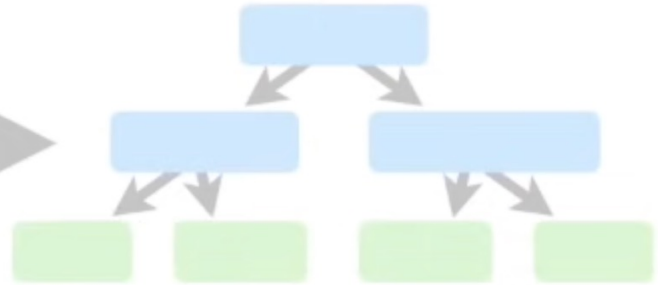
So let's start by plugging in **71.2** for the **Predicted Weight**…

(Observed Weight - Predicted Weight)

Kudos to Joshua Starmer!

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
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

The errors that the previous tree made are the differences between the **Observed Weights**

(**Observed Weight** - Predicted Weight)

Kudos to Joshua Starmer!

Average Weight

71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
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	

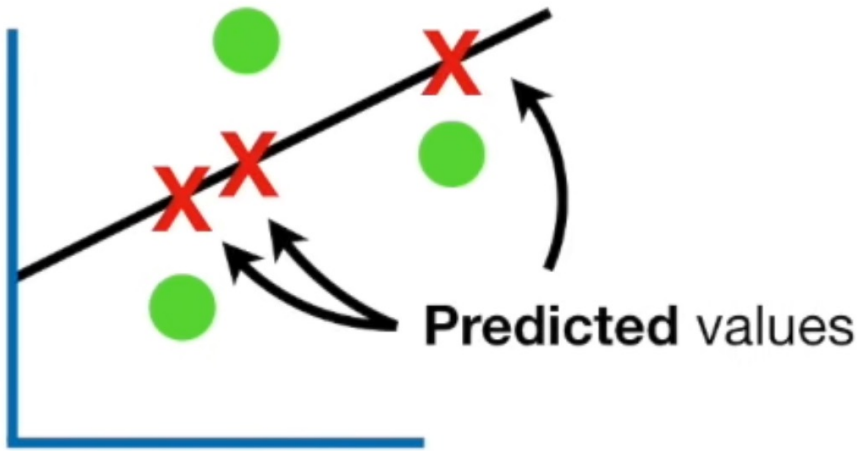
Kudos to Joshua Starmer!

Average Weight

71.2

NOTE: The term **Pseudo Residual** is based on **Linear Regression**, where the difference between the **Observed** values and the **Predicted** values results in **Residuals**.

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
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	

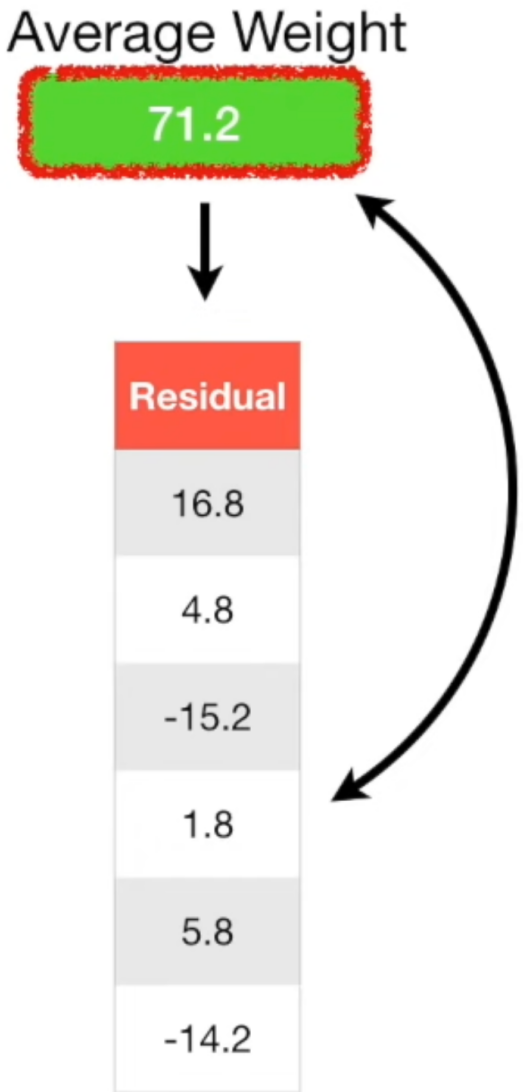


Kudos to Joshua Starmer!

Average Weight

71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
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	

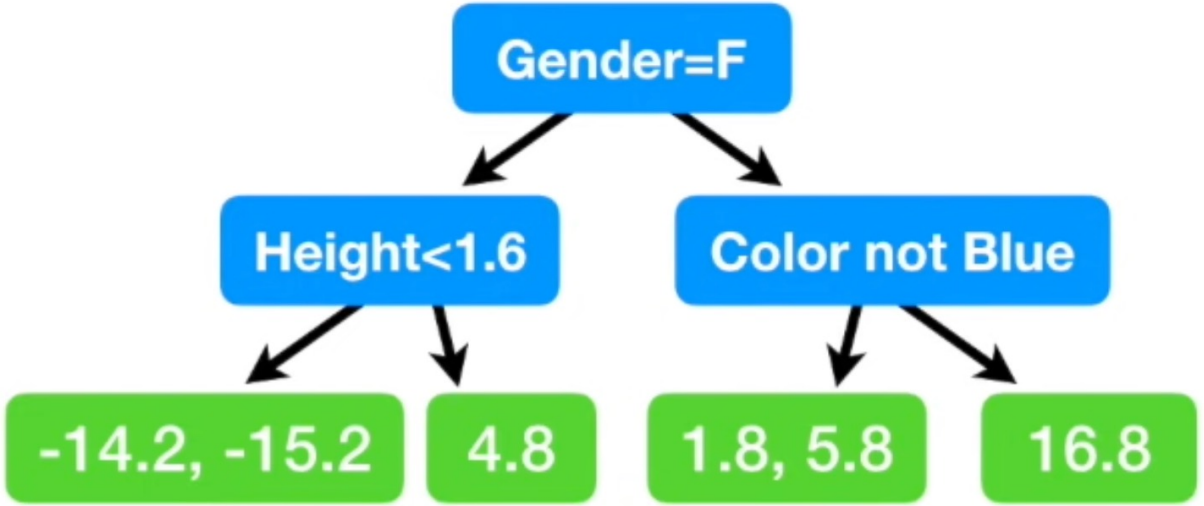


Kudos to Joshua Starmer!

After computing the residuals, we build a tree that aims to predict the residuals!

We'll do the same procedure later, different times...

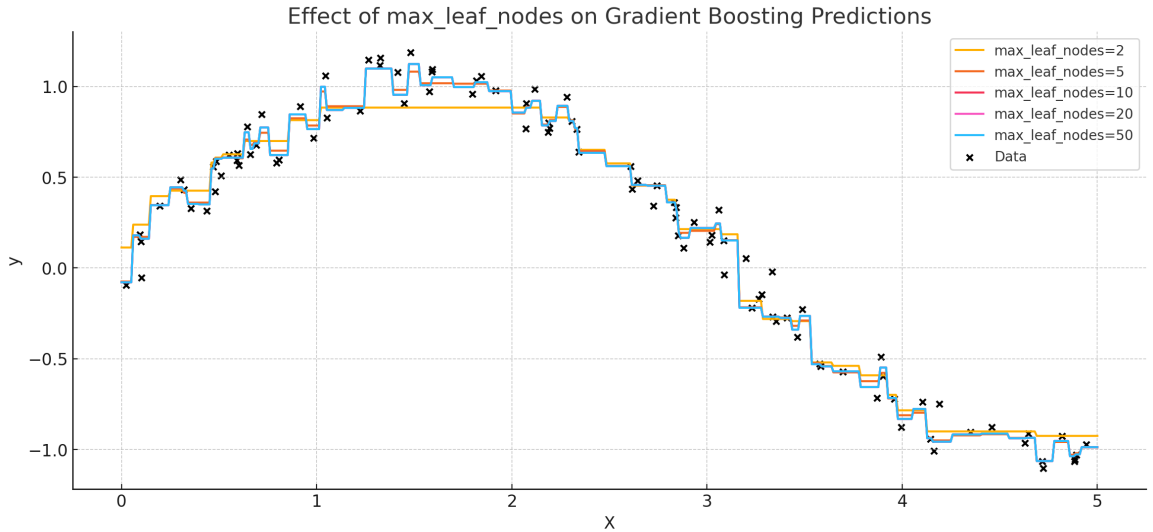
Height (m)	Favorite Color	Gender	Residual
1.6	Blue	Male	16.8
1.6	Green	Female	4.8
1.5	Blue	Female	-15.2
1.8	Red	Male	1.8
1.5	Green	Male	5.8
1.4	Blue	Female	-14.2

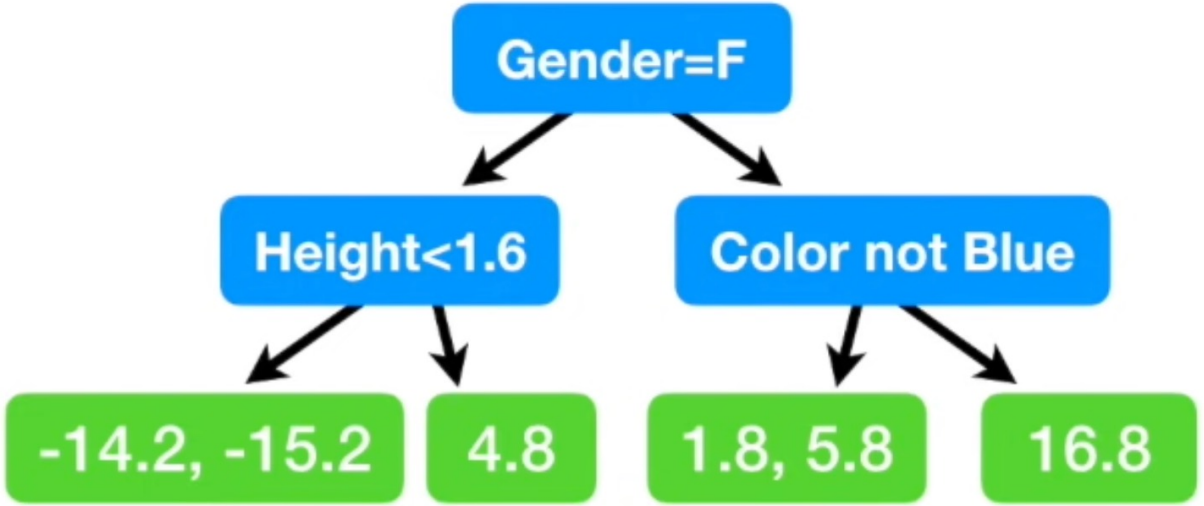


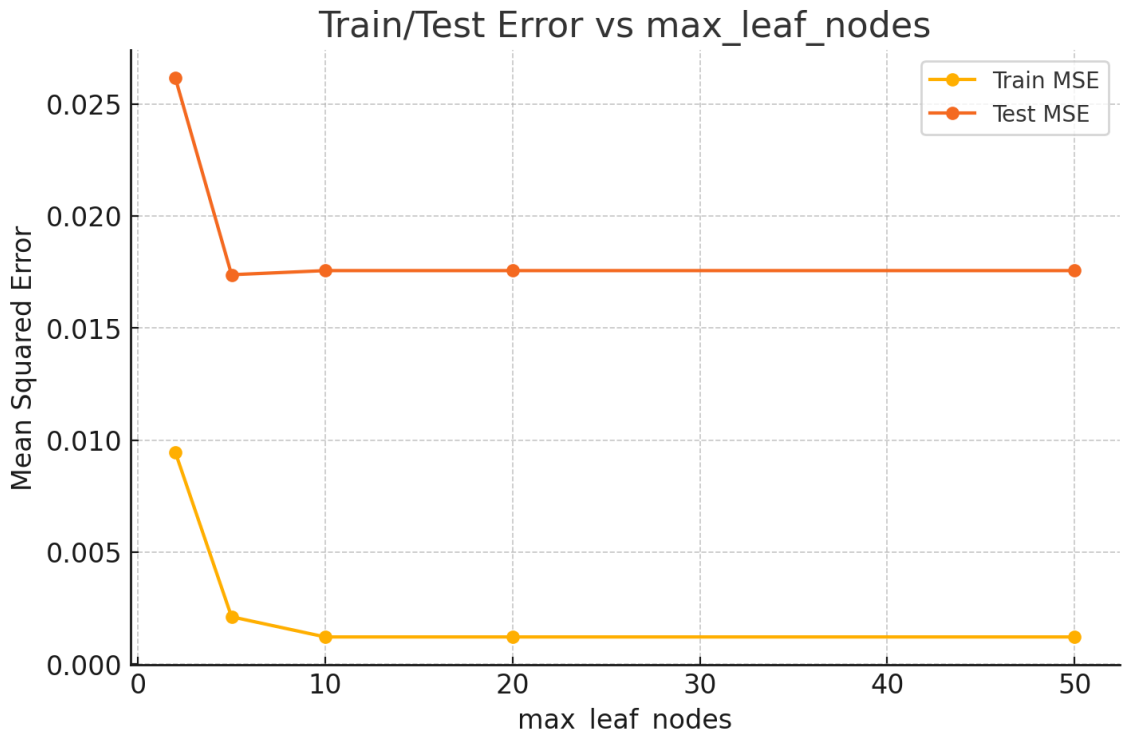
Our regression tree has a constrain: the max amount of leaves! In this example 4!

In practice, from 8 to 32

Kudos to Joshua Starmer!





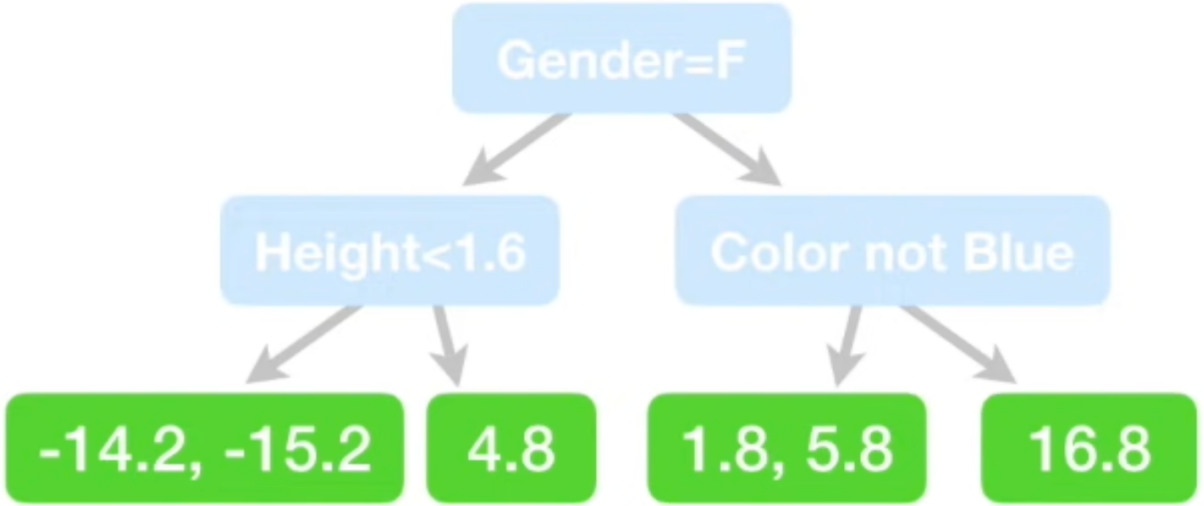


Our regression tree has a constrain: the max amount of leaves! In this example 4!

In practice, from 8 to 32

Kudos to Joshua Starmer!

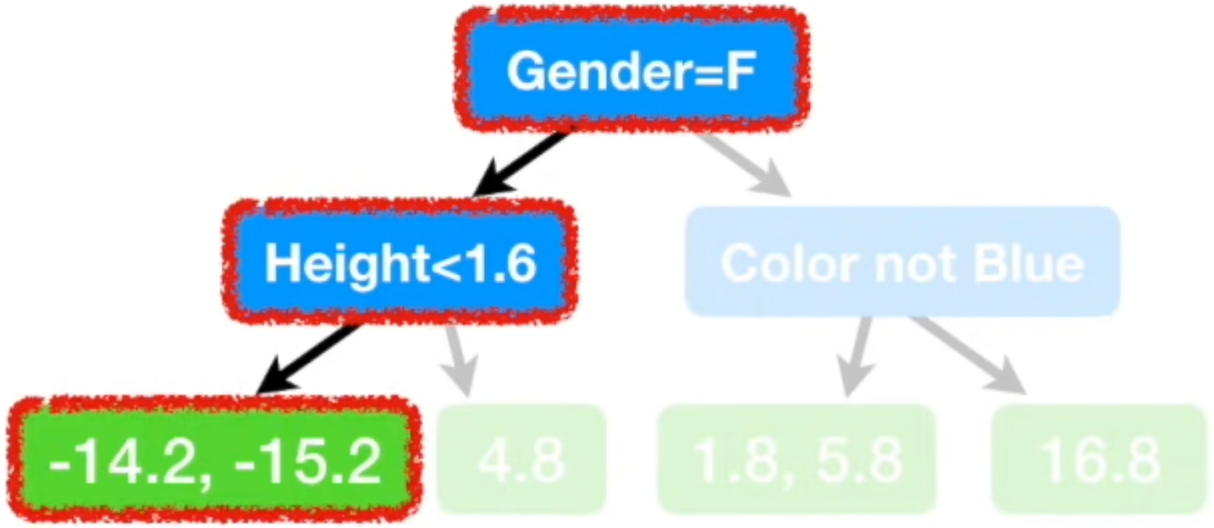
Height (m)	Favorite Color	Gender	Residual
1.6	Blue	Male	16.8
1.6	Green	Female	4.8
1.5	Blue	Female	-15.2
1.8	Red	Male	1.8
1.5	Green	Male	5.8
1.4	Blue	Female	-14.2



By restricting the total number of leaves, we get fewer leaves than **Residuals**.

Kudos to Joshua Starmer!

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	56	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

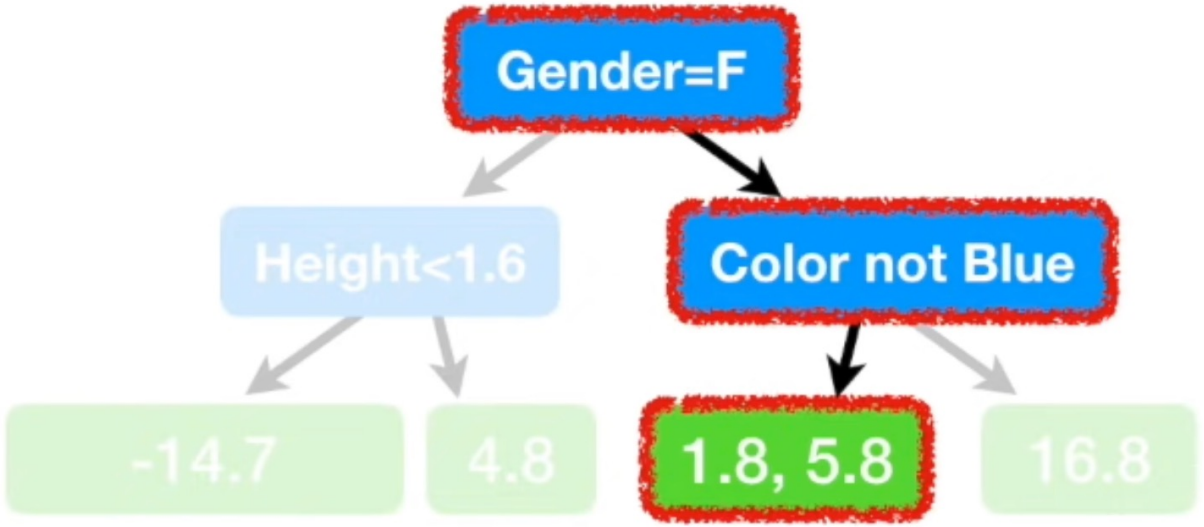


So we replace these residuals with their average.

$$\frac{(-14.2 + -15.2)}{2} = -14.7$$

Kudos to Joshua Starmer!

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



So we replace these residuals with their average.

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

Kudos to Joshua Starmer!

Predicted Weight = 71.2 + 16.8 = 88

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

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

Kudos to Joshua Starmer!

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.

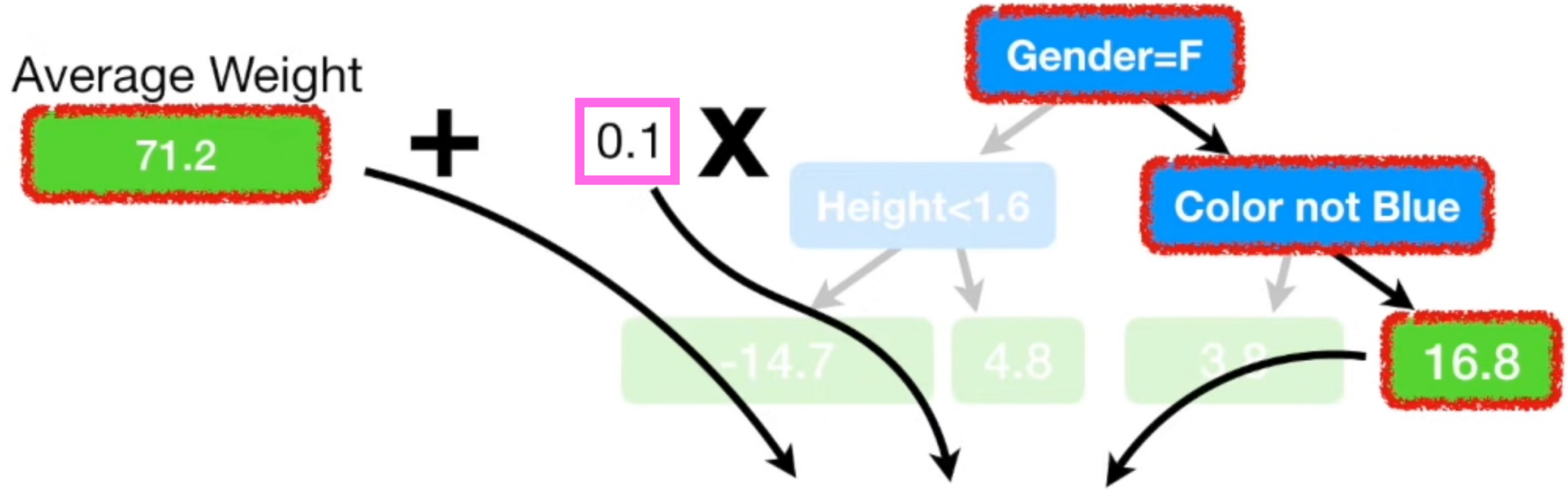
Kudos to Joshua Starmer!

Predicted Weight = 71.2 + 16.8 = 88

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

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

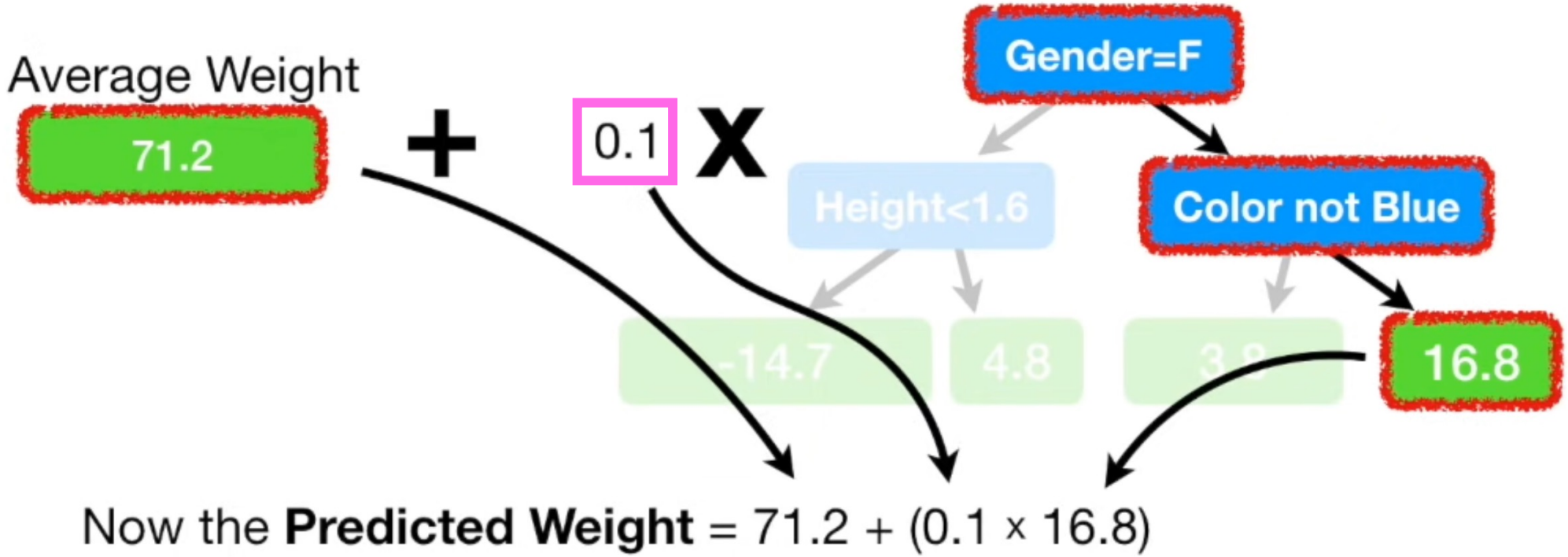
Kudos to Joshua Starmer!



Let's use only a 'small' part of the regression tree!
0.1 here is the learning rate!

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

Kudos to Joshua Starmer!



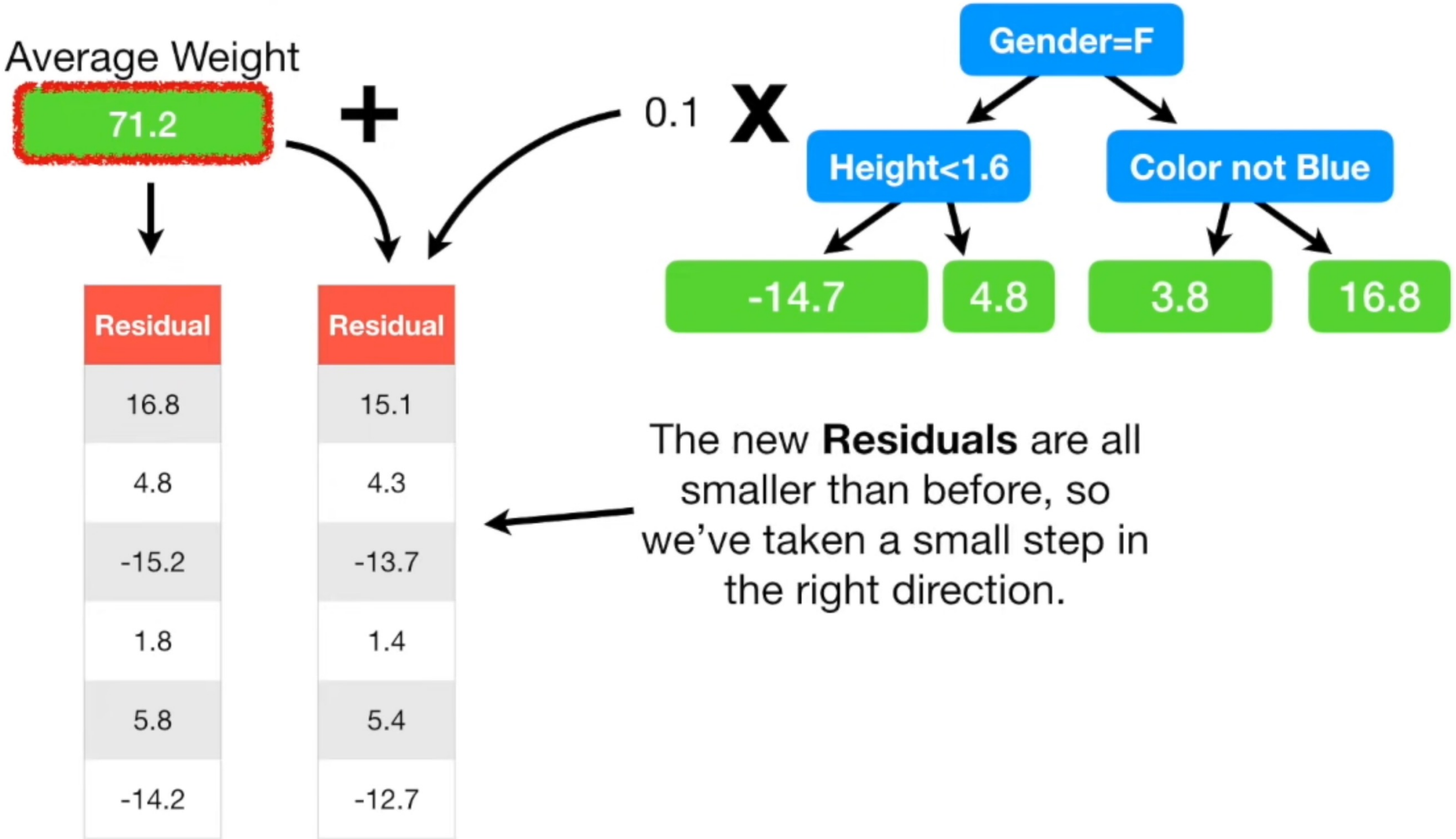
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

Kudos to Joshua Starmer!

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

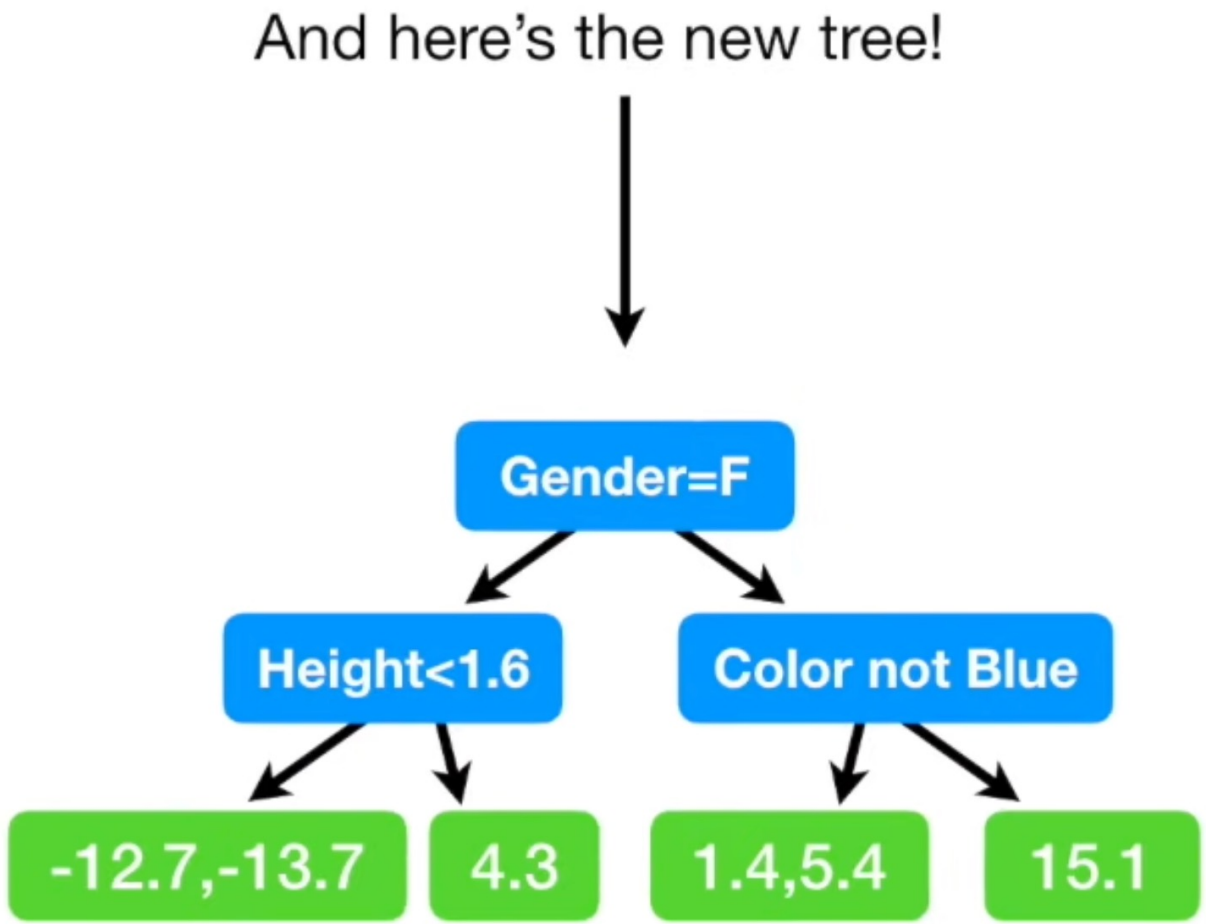
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

We are losing accuracy on training data with the hope to get better generalization (less overfitting and variance!)



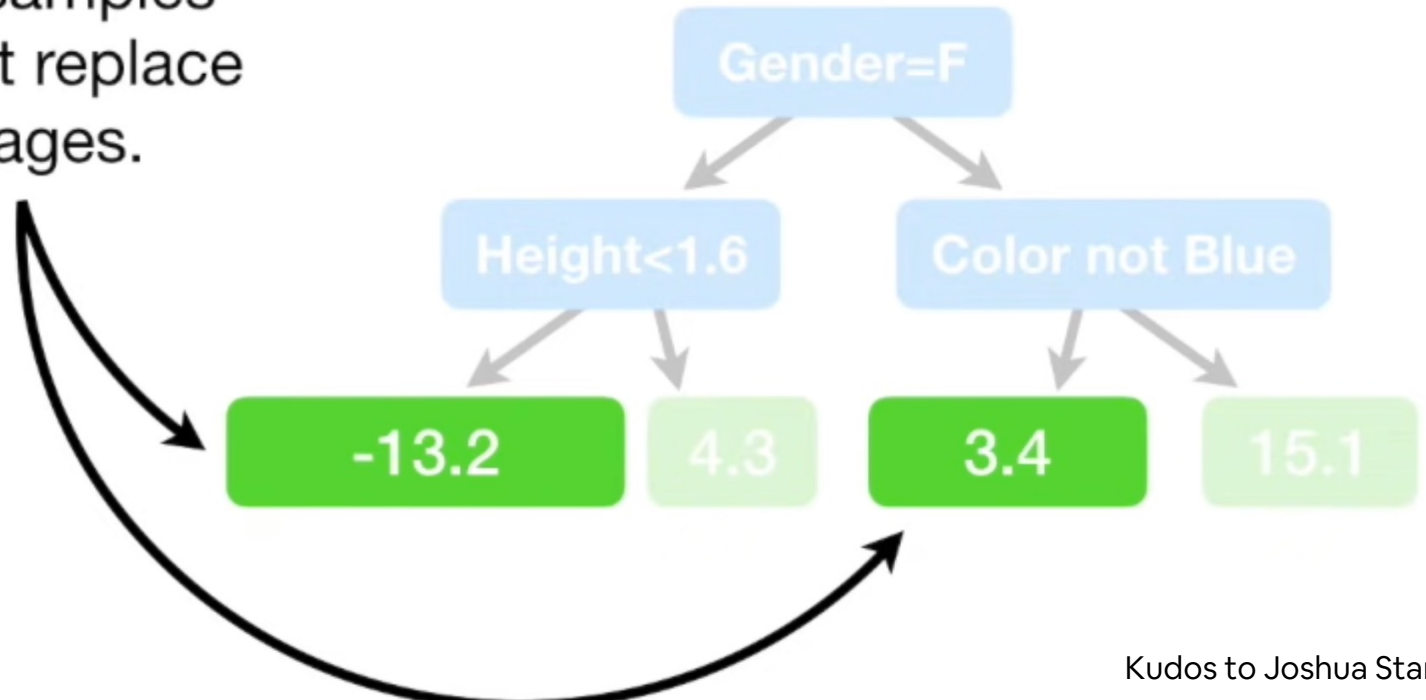
Kudos to Joshua Starmer!

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

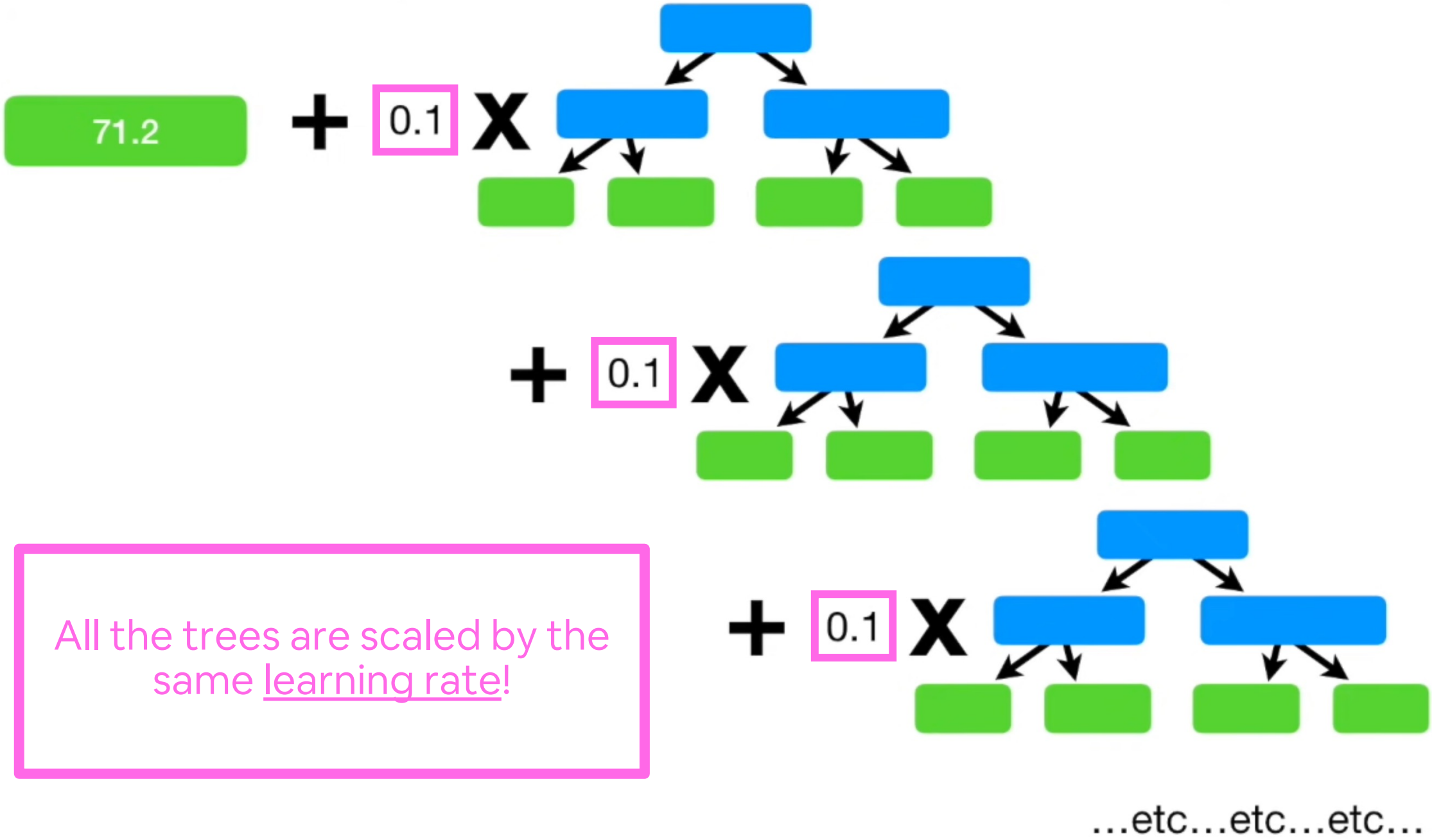


Kudos to Joshua Starmer!

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

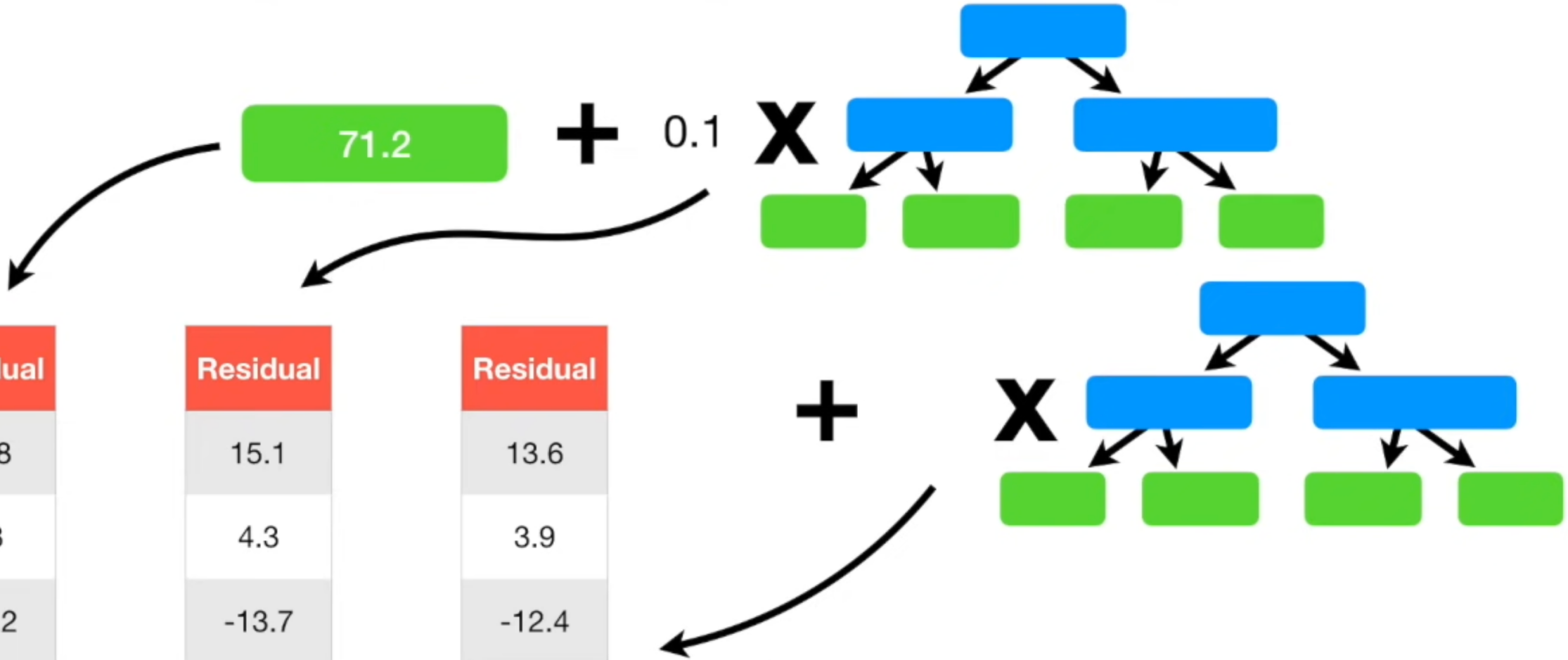


Kudos to Joshua Starmer!



All the trees are scaled by the same learning rate!

Kudos to Joshua Starmer!



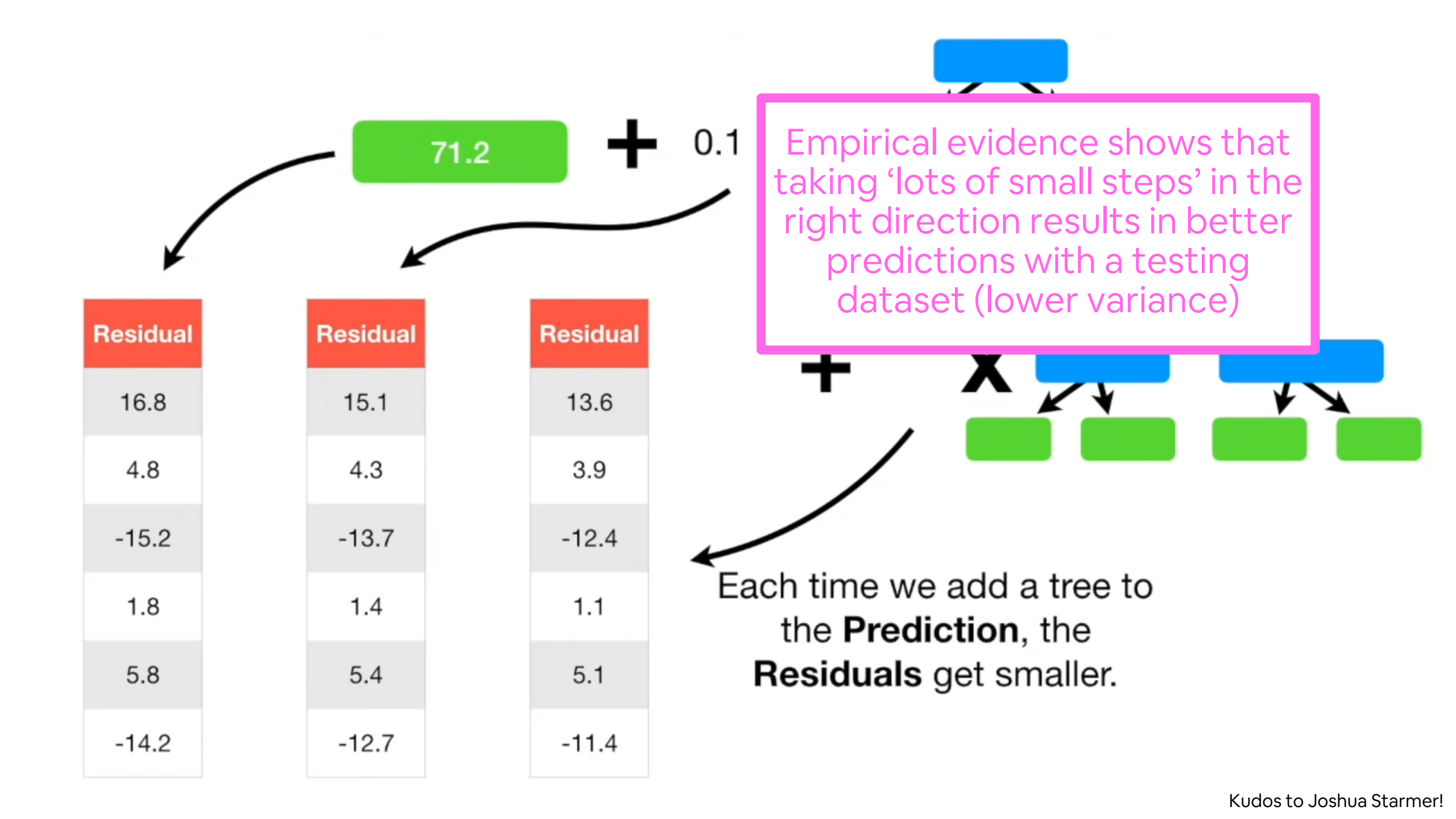
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2

Residual
15.1
4.3
-13.7
1.4
5.4
-12.7

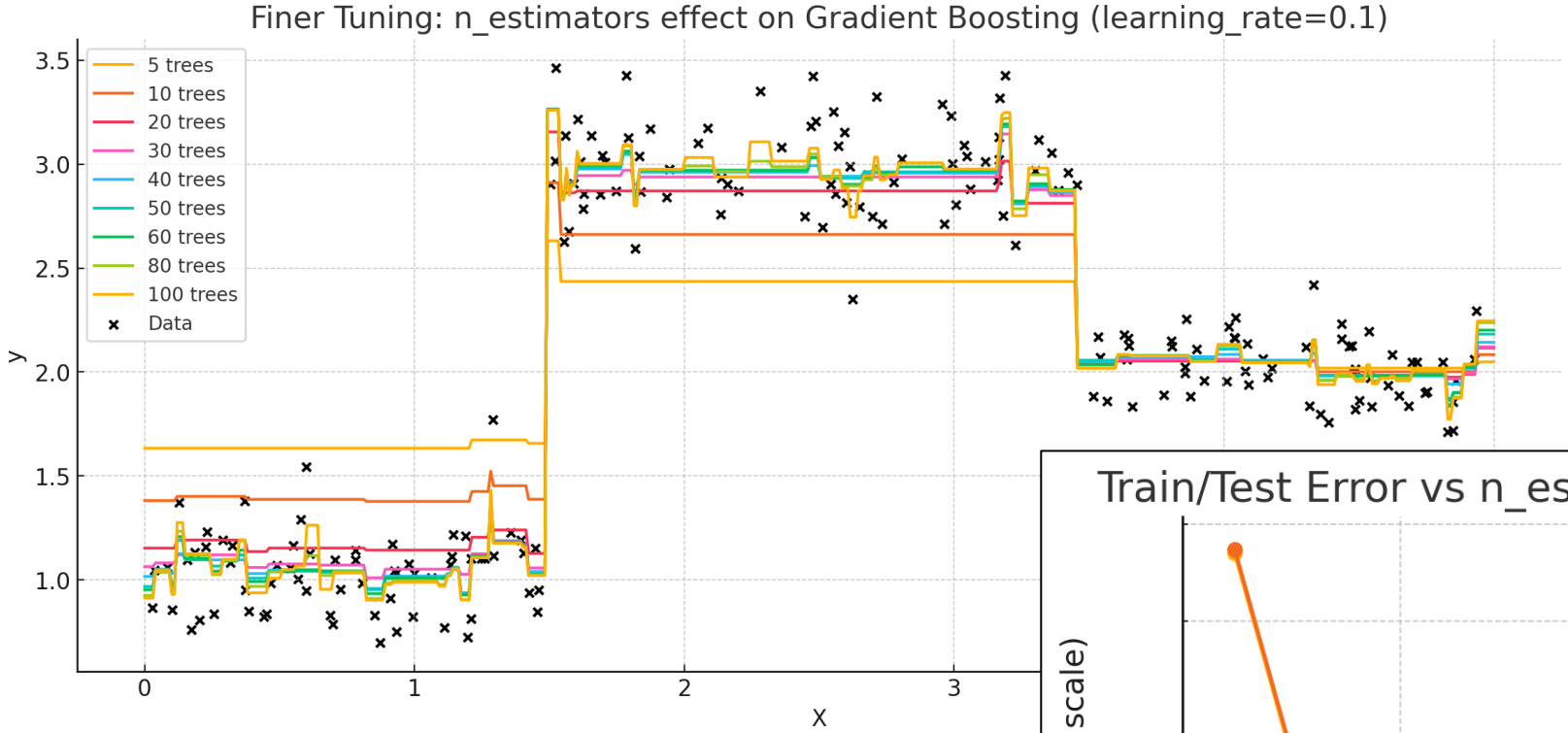
Residual
13.6
3.9
-12.4
1.1
5.1
-11.4

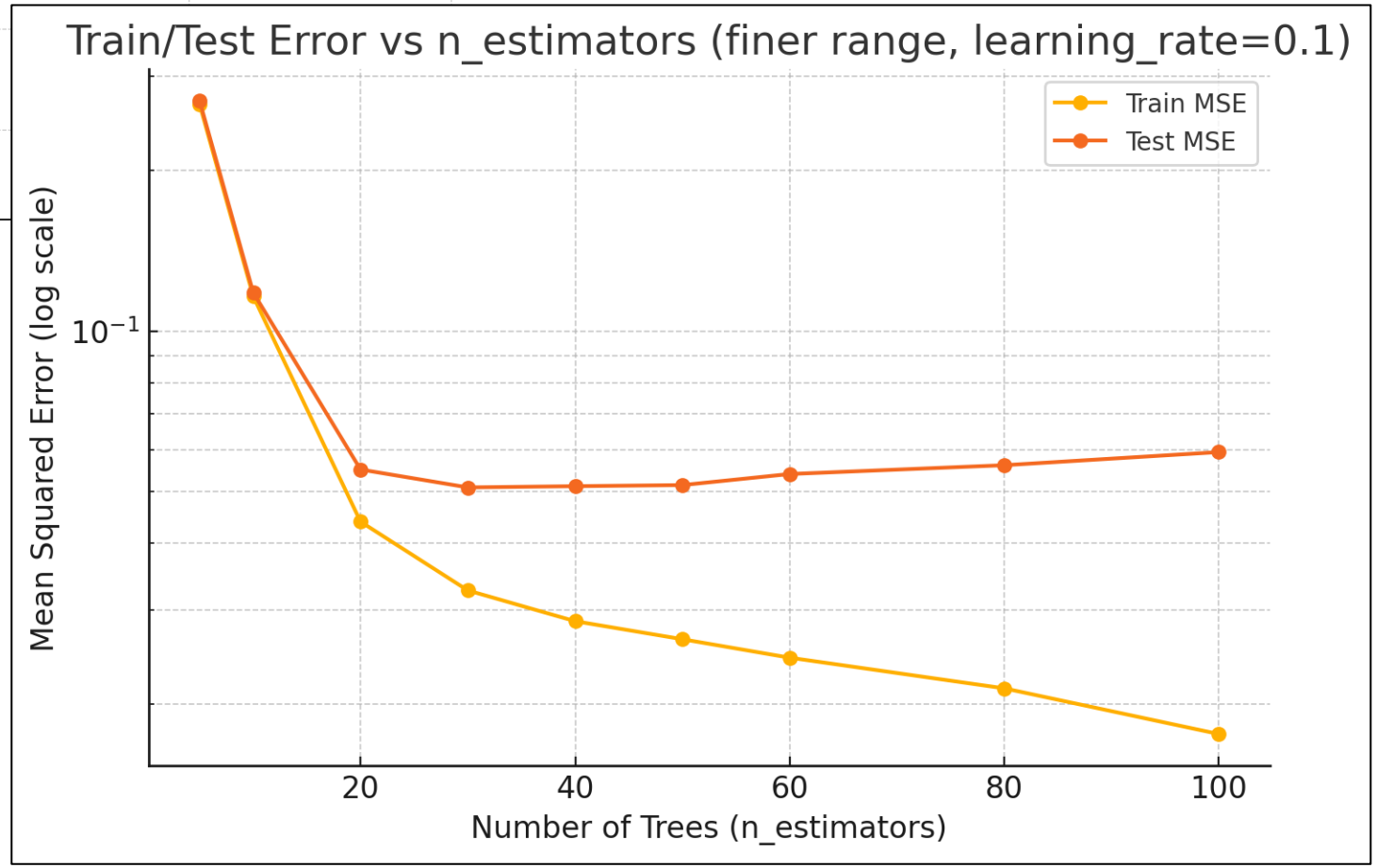
Each time we add a tree to the **Prediction**, the **Residuals** get smaller.

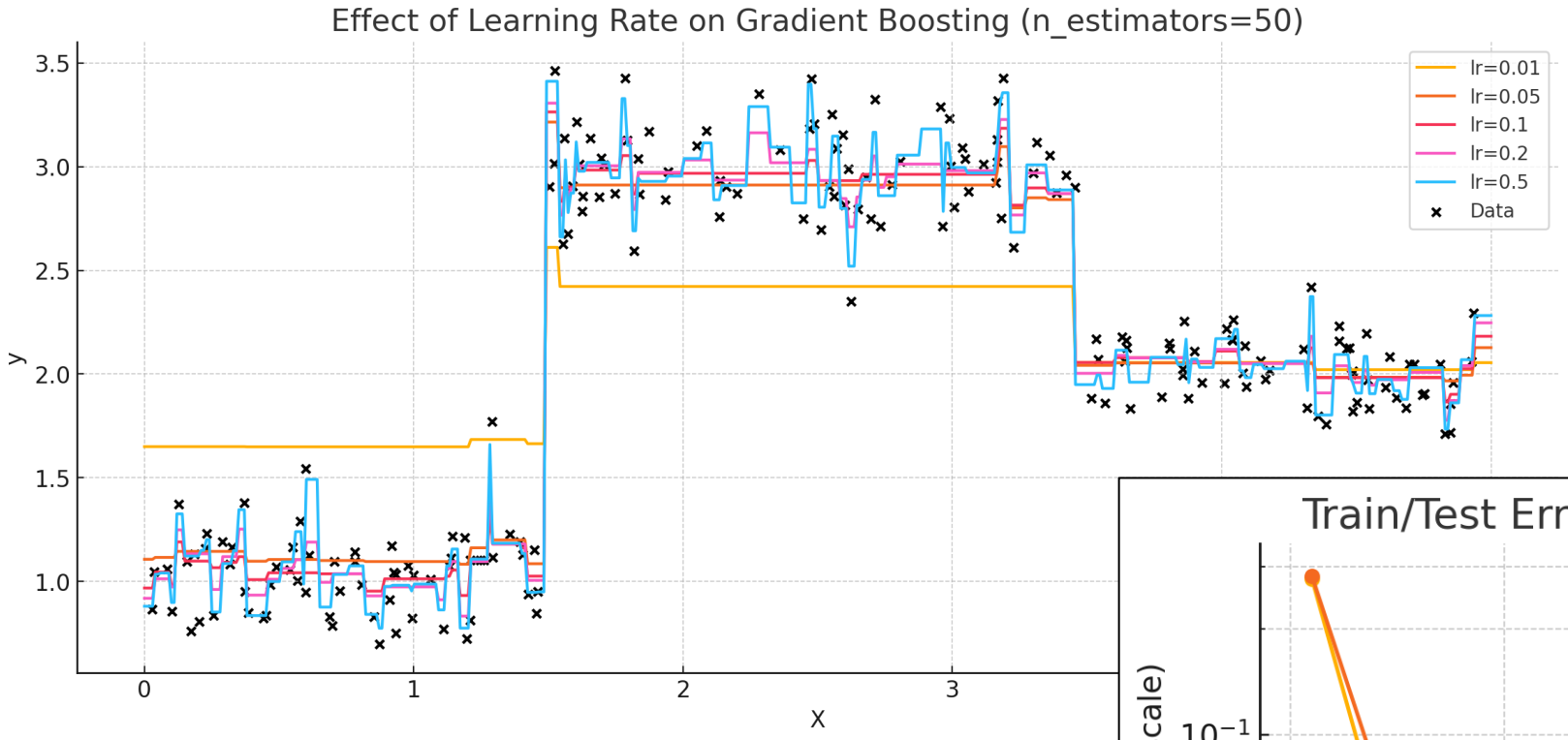
Kudos to Joshua Starmer!

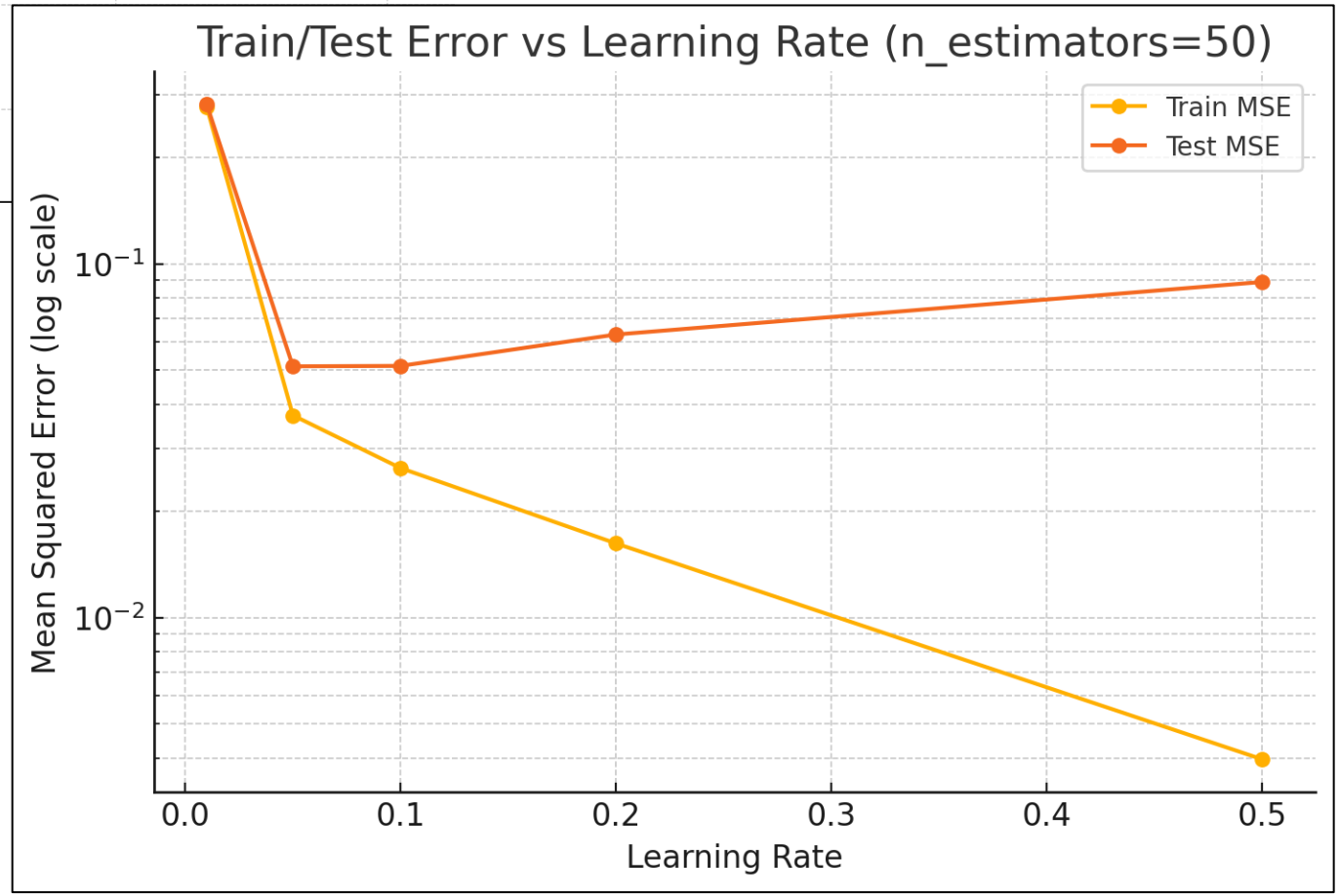


Kudos to Joshua Starmer!


Finer Tuning: n_estimators effect on Gradient Boosting (learning_rate=0.1)
5 trees
10 trees
20 trees
30 trees
40 trees
50 trees
60 trees
80 trees
100 trees
Data
y
X
3.5
3.0
2.5
2.0
1.5
1.0
0
1
2
3


Train/Test Error vs n_estimators (finer range, learning_rate=0.1)
Train MSE
Test MSE
Mean Squared Error (log scale)
10^{-1}
20
40
60
80
100
Number of Trees (n_estimators)


Effect of Learning Rate on Gradient Boosting (n_estimators=50)
lr=0.01
lr=0.05
lr=0.1
lr=0.2
lr=0.5
Data
y
X
3.5
3.0
2.5
2.0
1.5
1.0
0
1
2
3


Train/Test Error vs Learning Rate (n_estimators=50)
Train MSE
Test MSE
Mean Squared Error (log scale)
10^{-1}
10^{-2}
Learning Rate
0.0
0.1
0.2
0.3
0.4
0.5

Method	Core Idea	Model Combination	Key Traits
Bagging	Train trees on random data subsets	Averaging / Voting	Reduces variance, parallelizable, robust to overfitting
Random Forest	Bagging + random feature selection	Averaging / Voting	Strong baseline, good generalization
Boosting	Sequential models to fix previous errors	Weighted sum	Reduces bias, sensitive to noise
AdaBoost	Focus on misclassified samples via reweighting	Weighted sum	Simple, uses weak learners (e.g., stumps), effective on clean data
Gradient Boosting	Fit to loss function gradients	Weighted sum	Flexible loss functions, can overfit without tuning
XGBoost	Regularized GBM with pruning and optimizations	Weighted sum	Fast, regularized, handles missing values
LightGBM	Histogram-based GBM, leaf-wise growth	Weighted sum	Very fast, memory-efficient, great for large-scale problems
CatBoost	Categorical-feature-friendly GBM	Weighted sum	Handles categoricals natively, avoids overfitting
Stacked Ensemble	Combine diverse models with meta-learner	Meta-model (e.g., regression)	Very flexible, risk of overfitting without proper cross-validation

Adaboost is really similar!

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

…then **AdaBoost** starts by building a very short tree, called a **Stump**, from the **Training Data**.

Then **AdaBoost** builds the next stump based on errors that the previous stump made…

Only 2 leaves!

etc. etc. etc.

Two main differences:

1) AdaBoost only have stumps (2 leaves) as trees

2) AdaBoost weights differently the trees depending on the weights (Gradient boosting have all the trees scaled by the same factor)

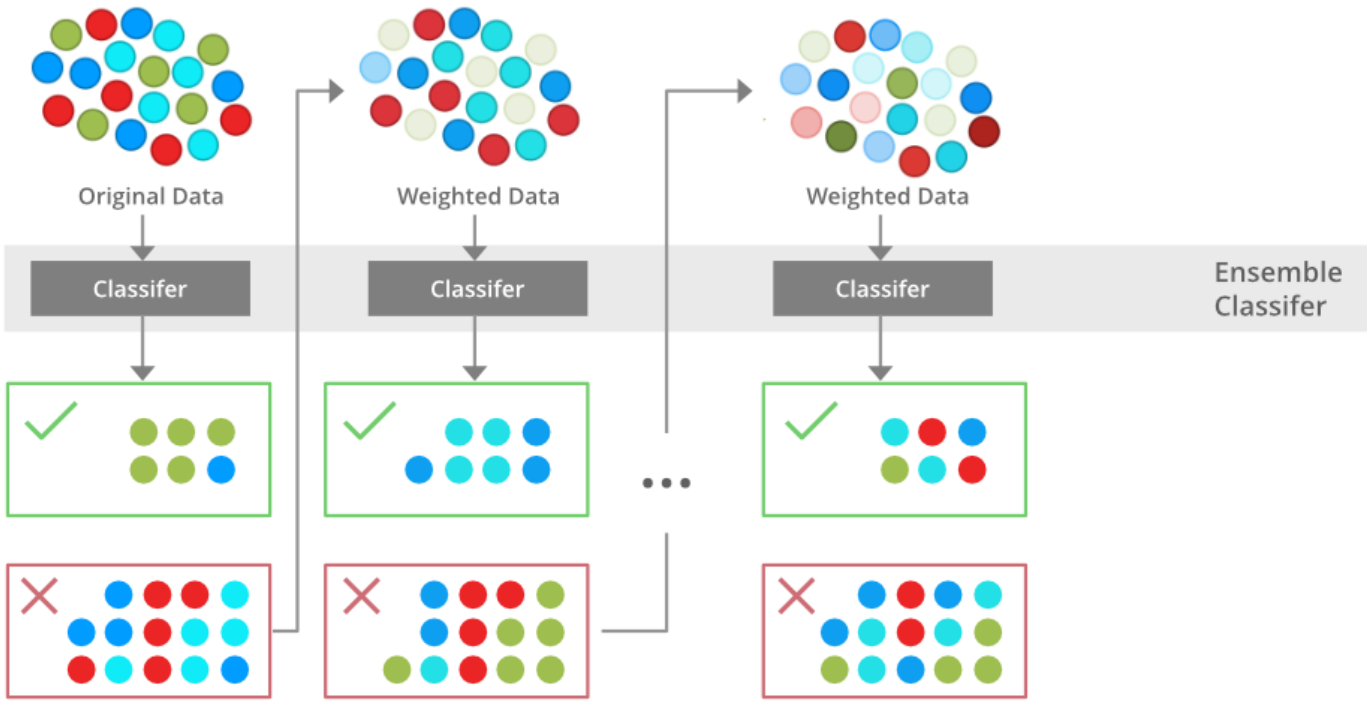
Huge kudos to Joshua Starmer!

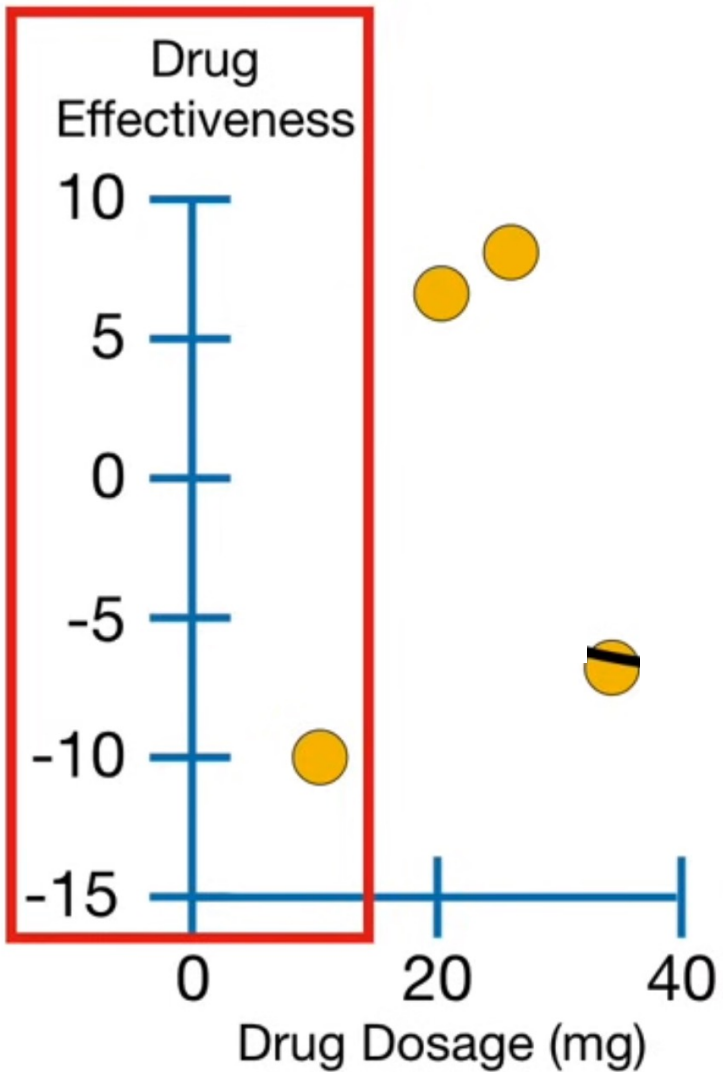
Method	Core Idea	Model Combination	Key Traits
Bagging	Train trees on random data subsets	Averaging / Voting	Reduces variance, parallelizable, robust to overfitting
Random Forest	Bagging + random feature selection	Averaging / Voting	Strong baseline, good generalization
Boosting	Sequential models to fix previous errors	Weighted sum	Reduces bias, sensitive to noise
AdaBoost	Focus on misclassified samples via reweighting	Weighted sum	Simple, uses weak learners (e.g., stumps), effective on clean data
Gradient Boosting	Fit to loss function gradients	Weighted sum	Flexible loss functions, can overfit without tuning
XGBoost	Regularized GBM with pruning and optimizations	Weighted sum	Fast, regularized, handles missing values
LightGBM	Histogram-based GBM, leaf-wise growth	Weighted sum	Very fast, memory-efficient, great for large-scale problems
CatBoost	Categorical-feature-friendly GBM	Weighted sum	Handles categoricals natively, avoids overfitting
Stacked Ensemble	Combine diverse models with meta-learner	Meta-model (e.g., regression)	Very flexible, risk of overfitting without proper cross-validation

XGBoost

- A state-of-the art approach!
- Full of implementation tricks: we are just scratching the surface!
- It exploits:

1) Boosting (sequential trees)
2) A new way to construct trees, still based on the residuals
3) Regularization!

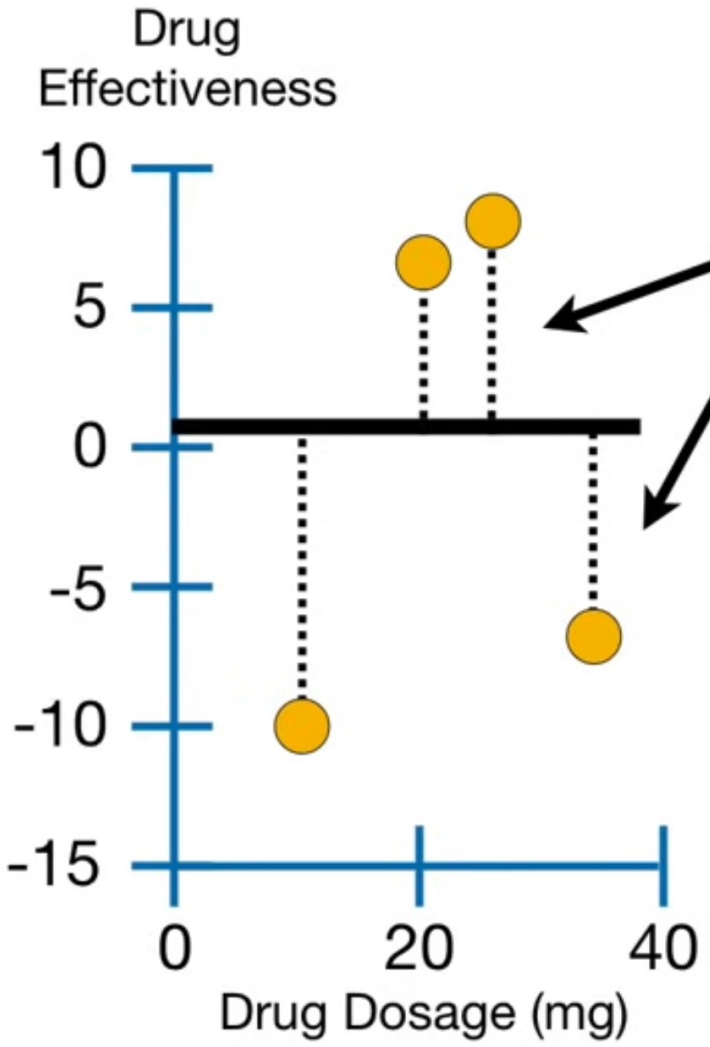




Kudos to Joshua Starmer!

0.5

First prediction: the mean! Black thick line...

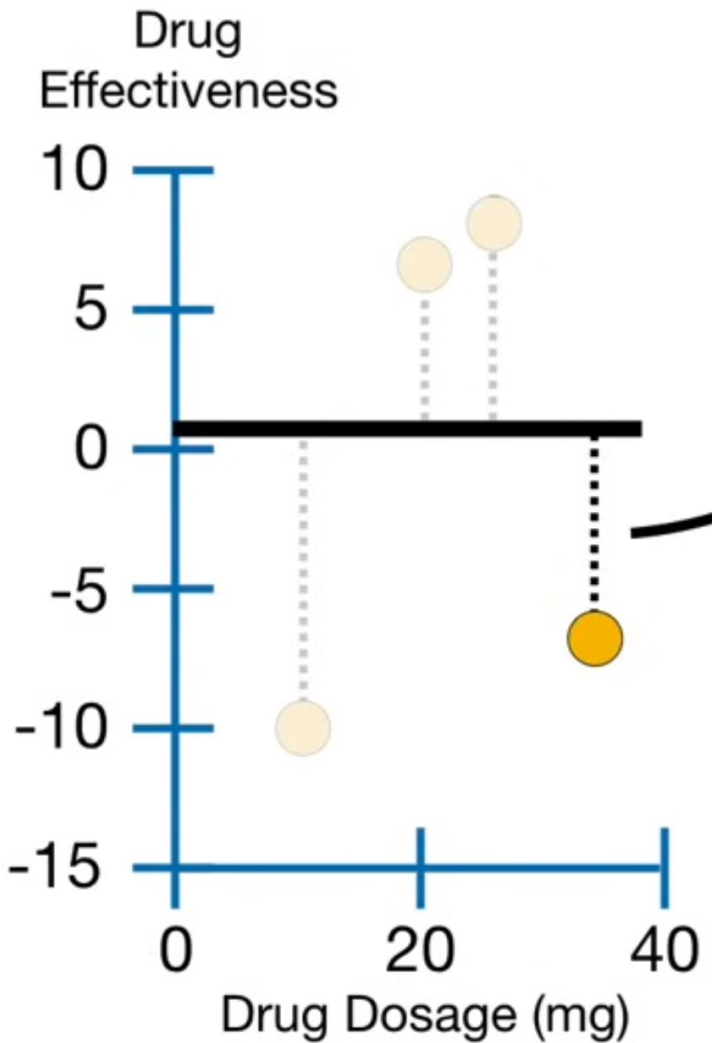


...and the **Residuals**, the differences between the **Observed** and **Predicted** values, show us how good the initial prediction is.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5



-10.5, 6.5, 7.5, -7.5

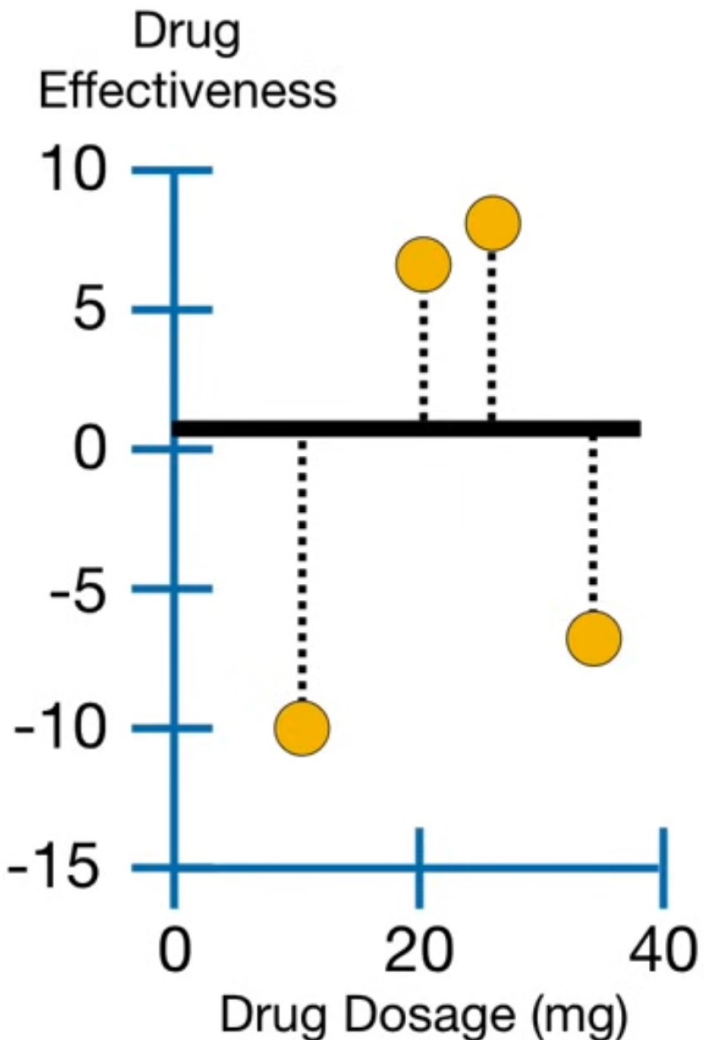
Trees here are not the standard regression trees: we also build them starting from residuals

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5

-10.5, 6.5, 7.5, -7.5



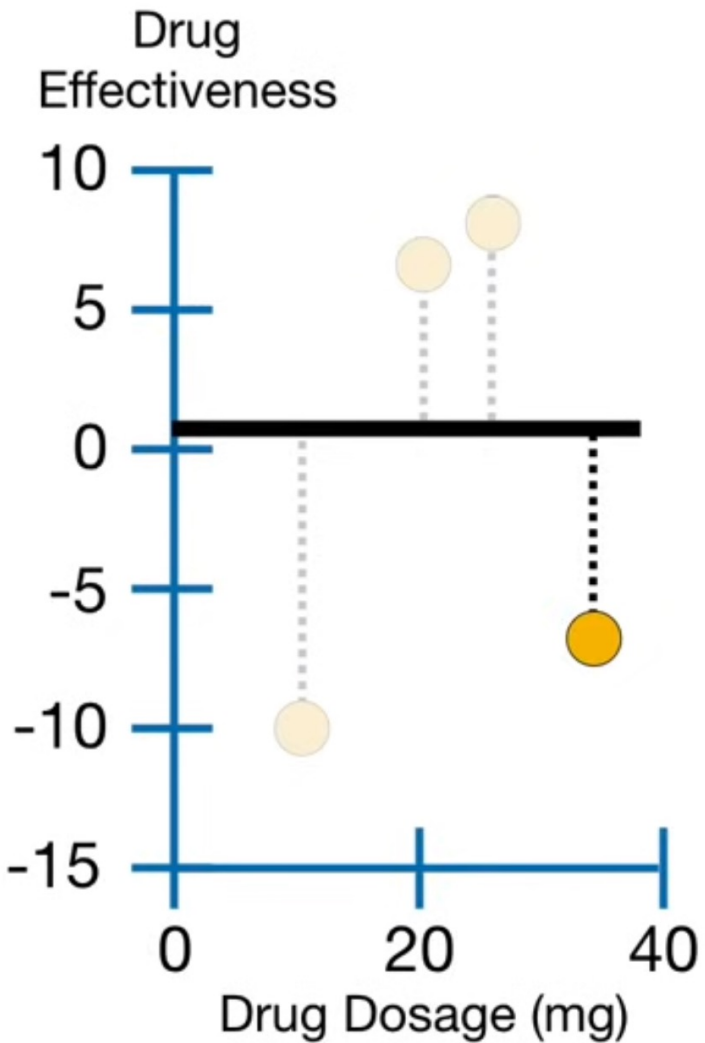
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \lambda}$$

NOTE: λ (lambda) is a **Regularization** parameter, and we'll talk more about that later.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5



-10.5, 6.5, 7.5, -7.5

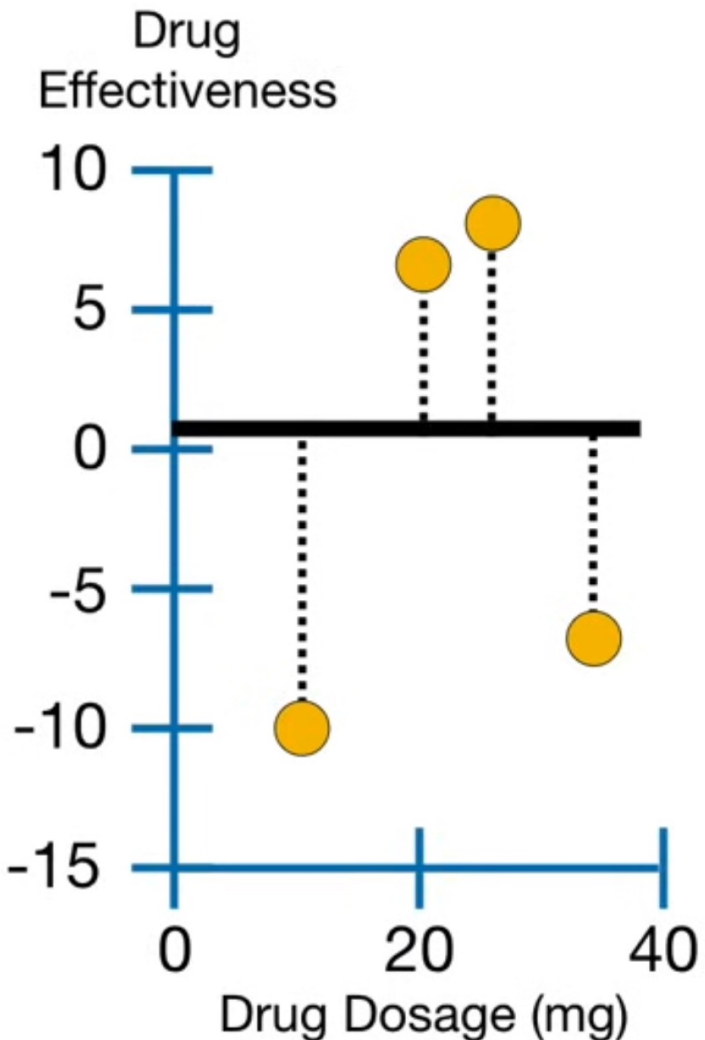
$$\text{Similarity Score} = \frac{(-10.5 + 6.5 + 7.5 + -7.5)^2}{\text{Number of Residuals} + \mathbf{0}} = 4$$

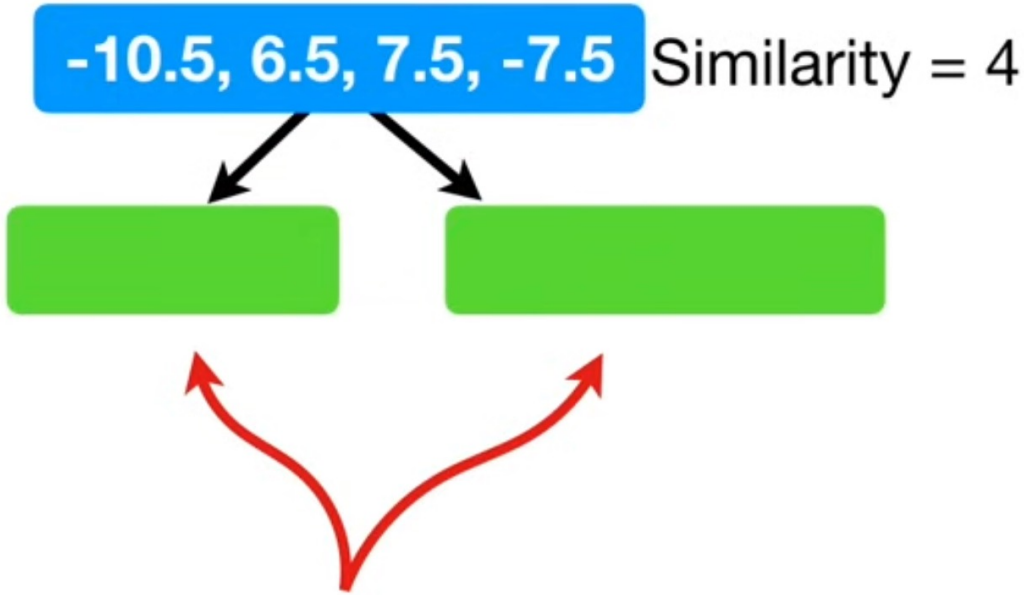
…and since there are **4** **Residuals** in the leaf, we put a **4** in the denominator.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5



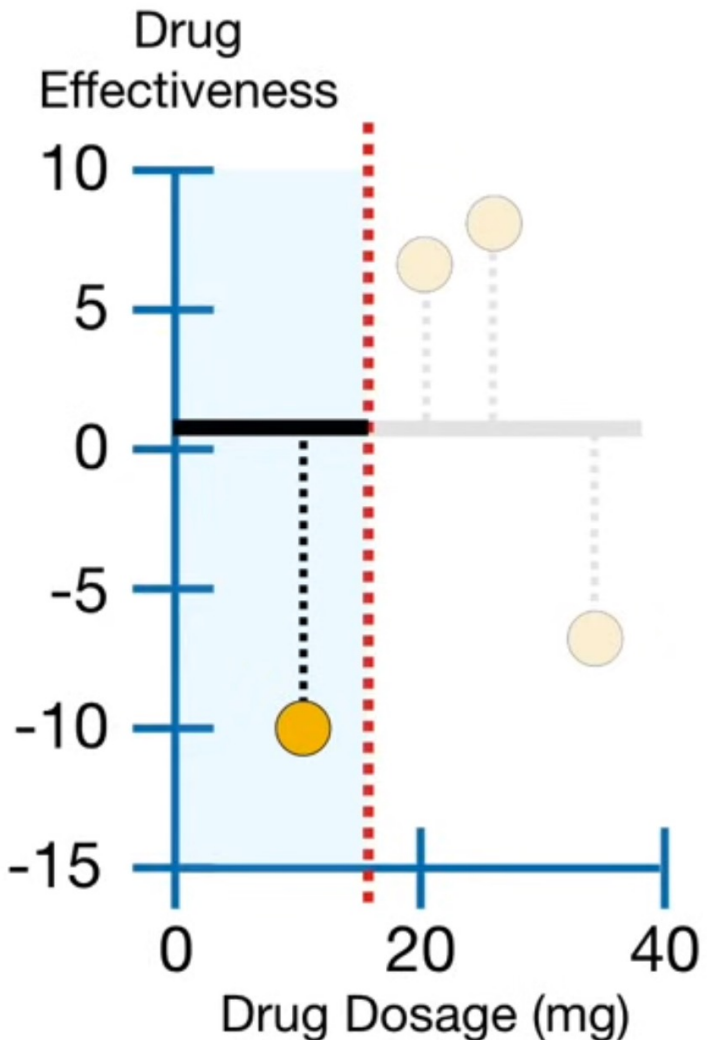


Now the question is whether or not we can do a better job clustering similar **Residuals** if we split them into two groups.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5



$$\text{Similarity Score} = \frac{-10.5^2}{\text{Number of Residuals} + \lambda}$$

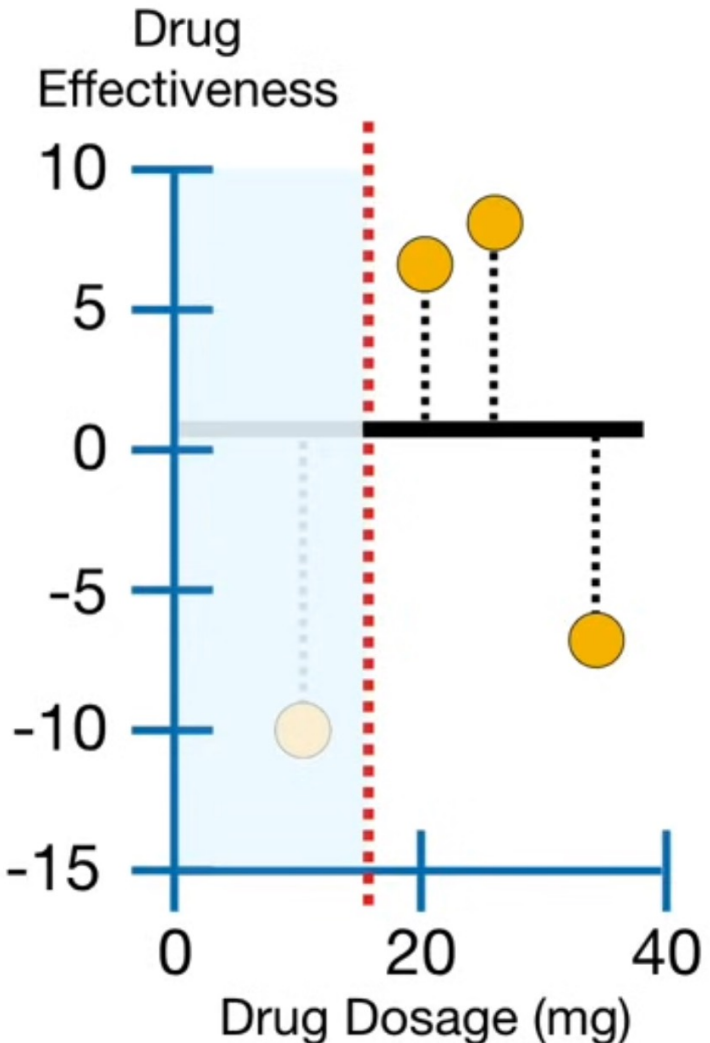
$$= 110.25$$

…and since only one **Residual** went to the leaf on the left, the **Number of Residuals = 1**.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

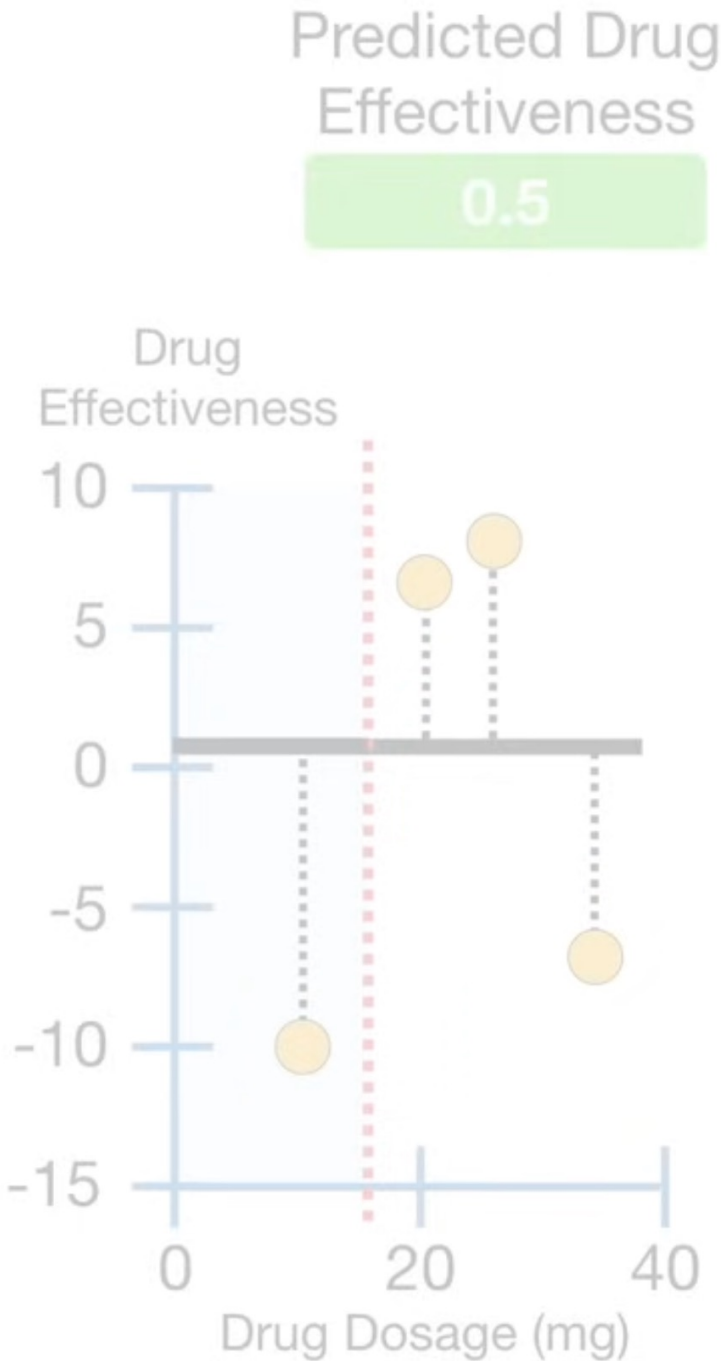
0.5

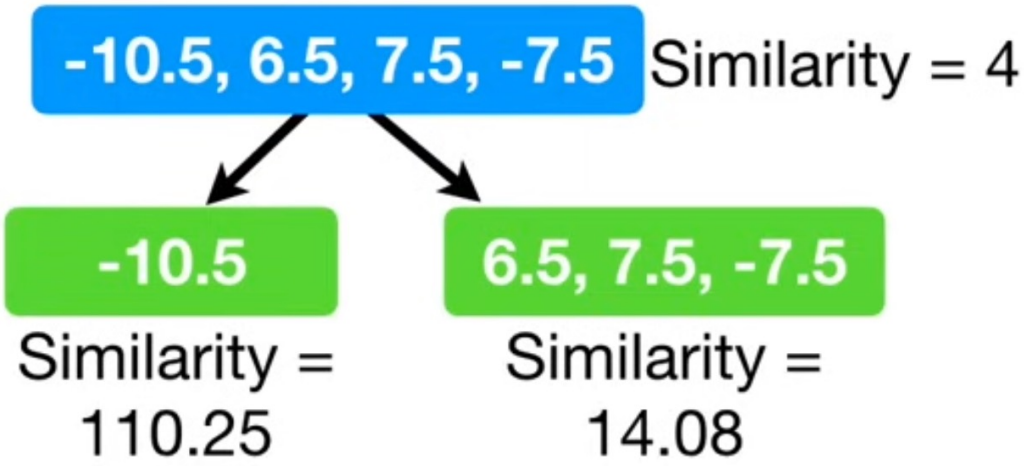


$$\text{Similarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{3 + \lambda}$$

…and just like before, let's let $\lambda = 0$.

Kudos to Joshua Starmer!





Gain = 110.25 + 14.08 - 4 = 120.33

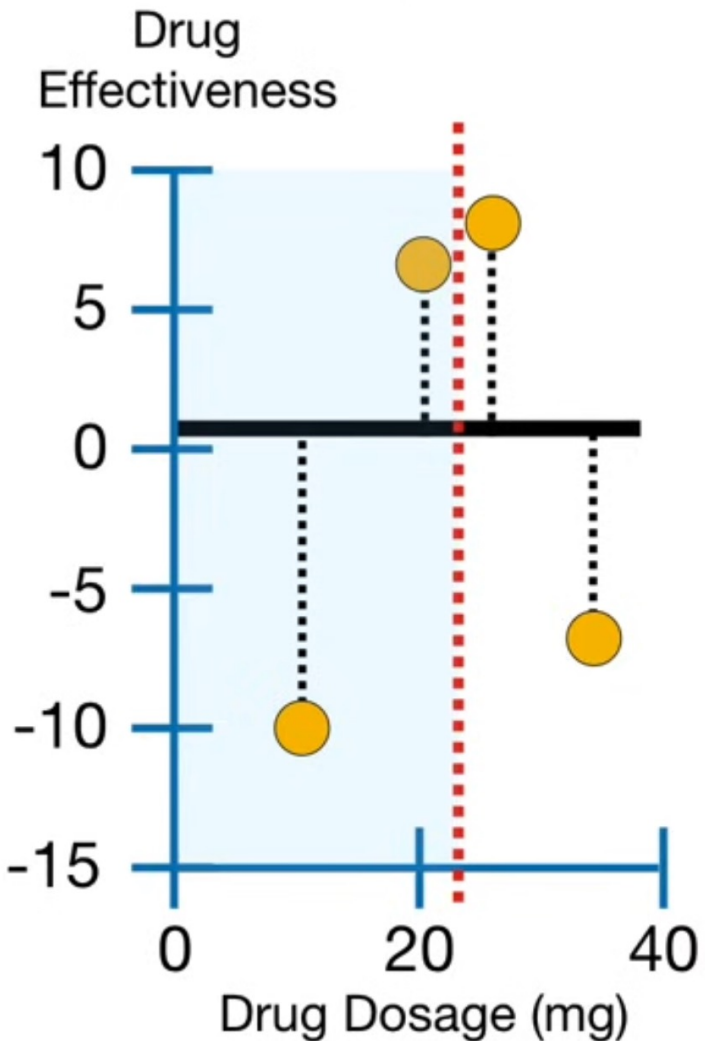
…gives us **120.33**.

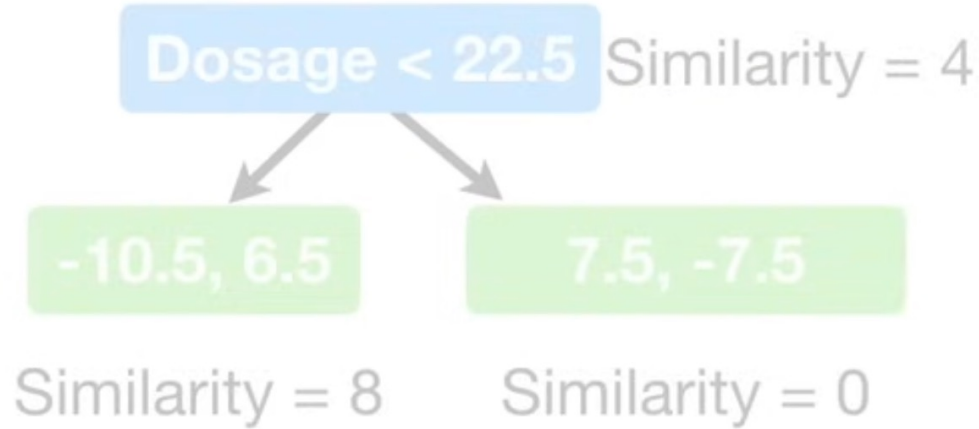
$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5





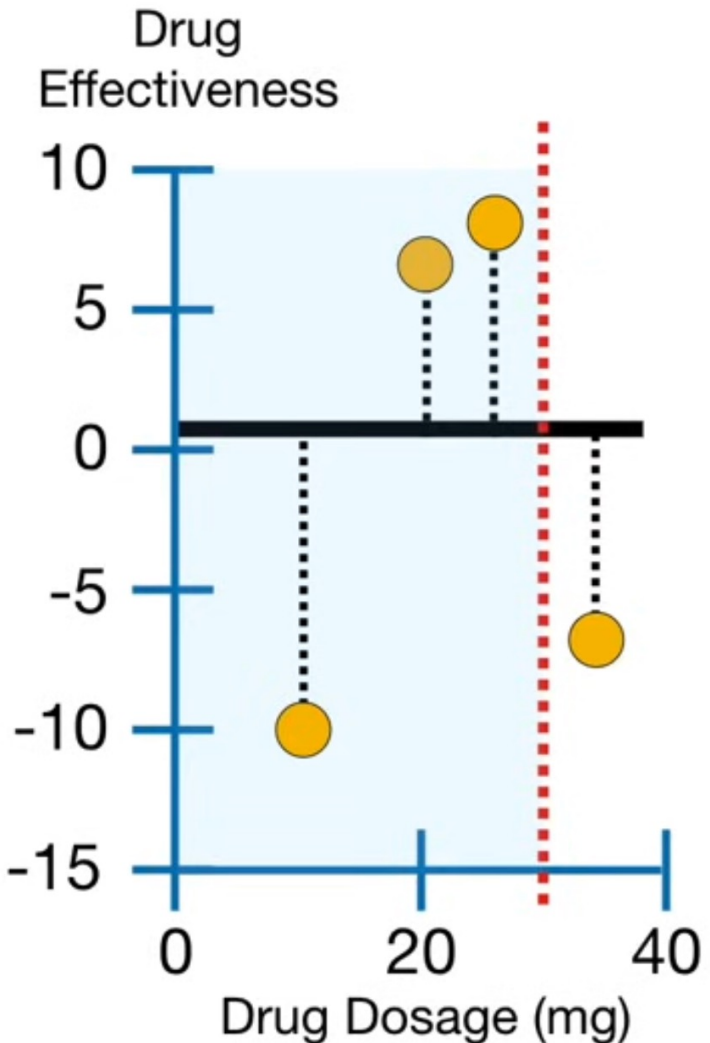
$$\text{Gain} = 8 + 0 - 4 = 4$$

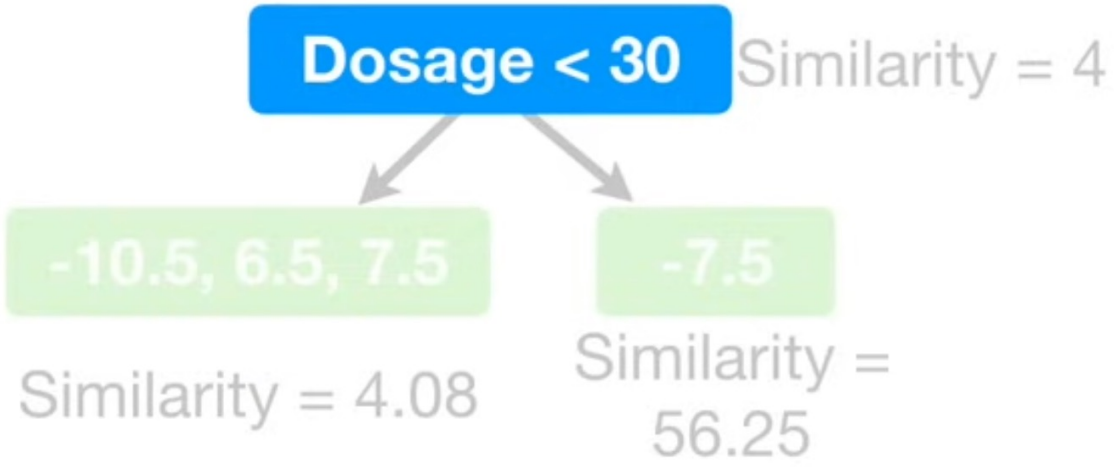
Since the **Gain** for **Dosage < 22.5** (**Gain = 4**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the **Residuals** into clusters of similar values.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5





Gain = 4.08 + 56.25 - 4 = 56.33

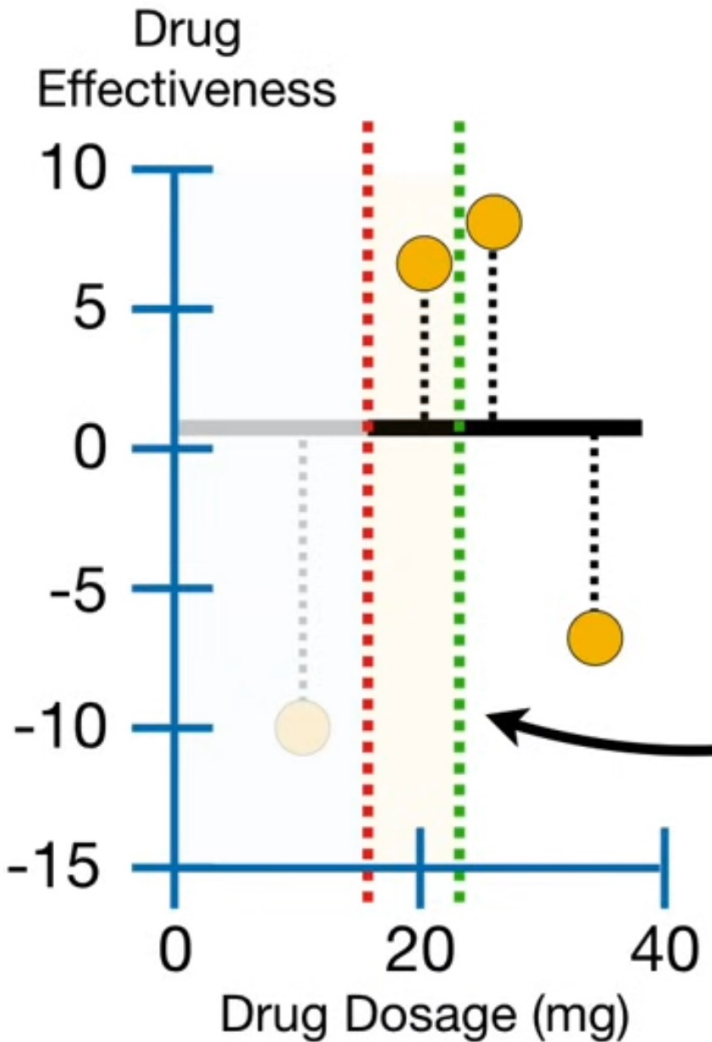
Again, since the **Gain** for **Dosage < 30** (**Gain = 56.33**) is less than the **Gain** for **Dosage < 15** (**Gain = 120.33**), **Dosage < 15** is better at splitting the observations.

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5

Best split!

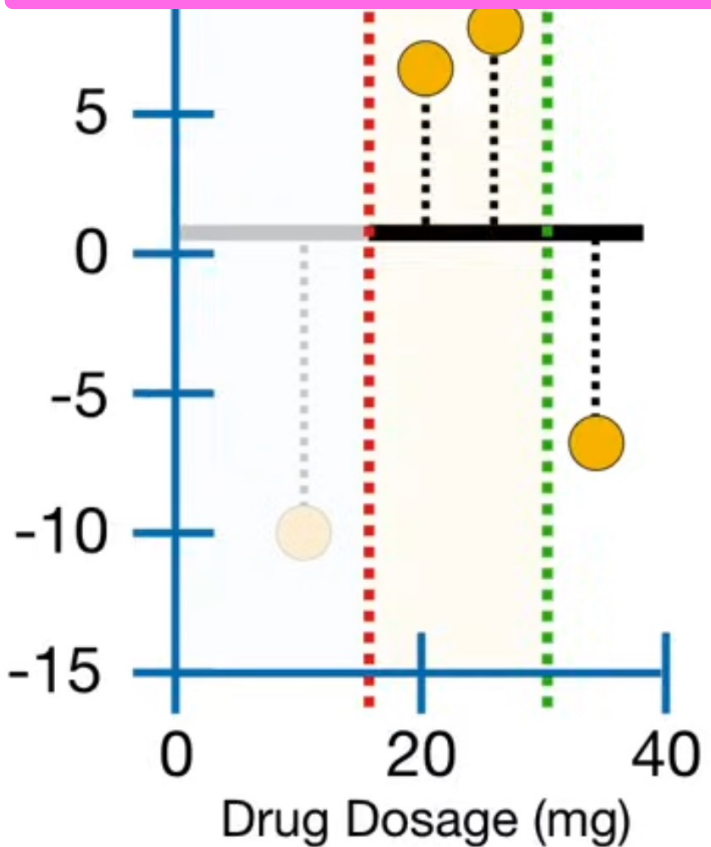


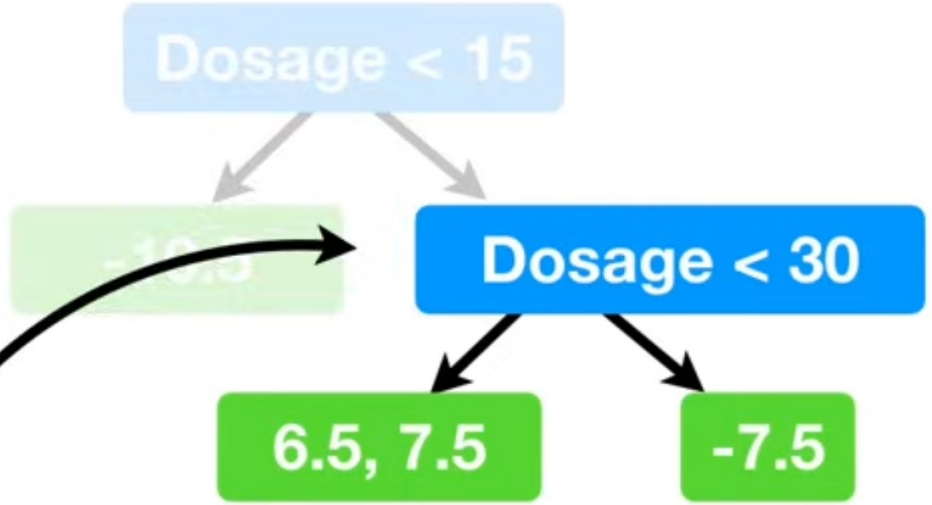
…and their average **Dosage** is **22.5**, which corresponds to this **dotted green line**.

Kudos to Joshua Starmer!

We do the same thing (looking at all splits and considering similarity score as a metric)

We decide to stop at depth = 2 in this example, but in reality, typically depth = 6

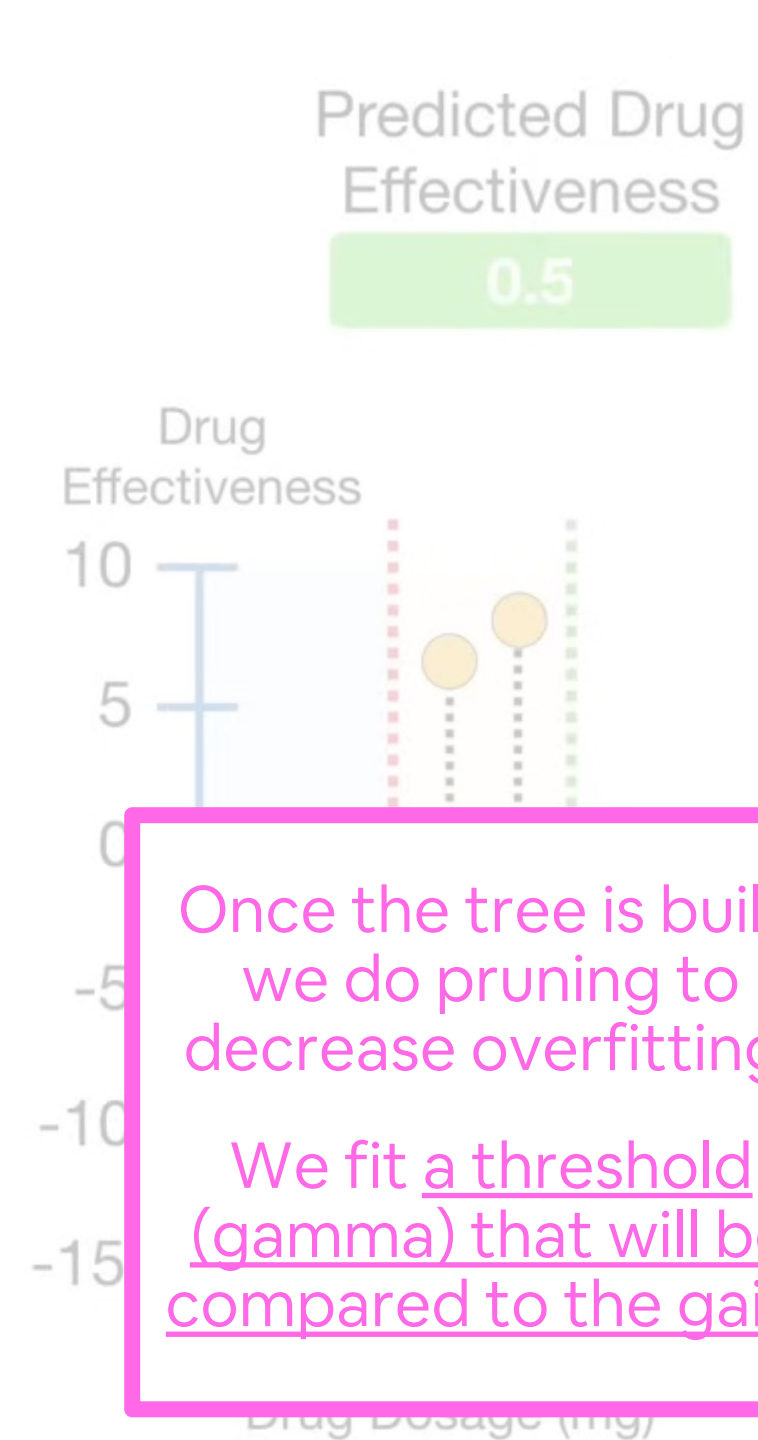




So we will use **Dosage < 30** as the threshold for this branch.

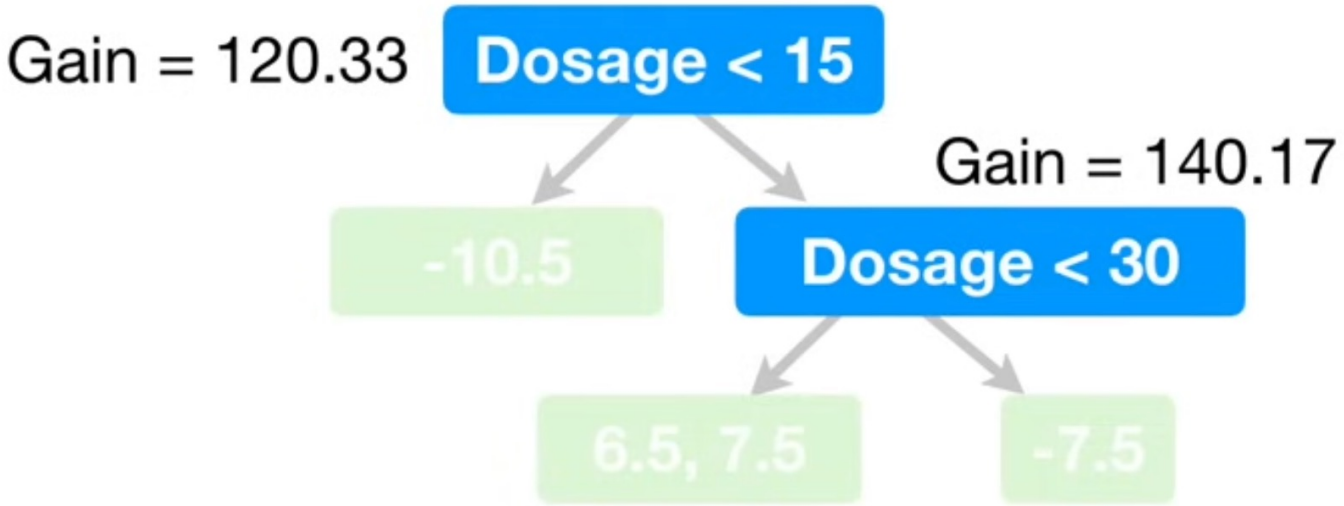
Gain = 98 + 56.25 - 14.08 = 140.17

Kudos to Joshua Starmer!



Once the tree is built, we do pruning to decrease overfitting!

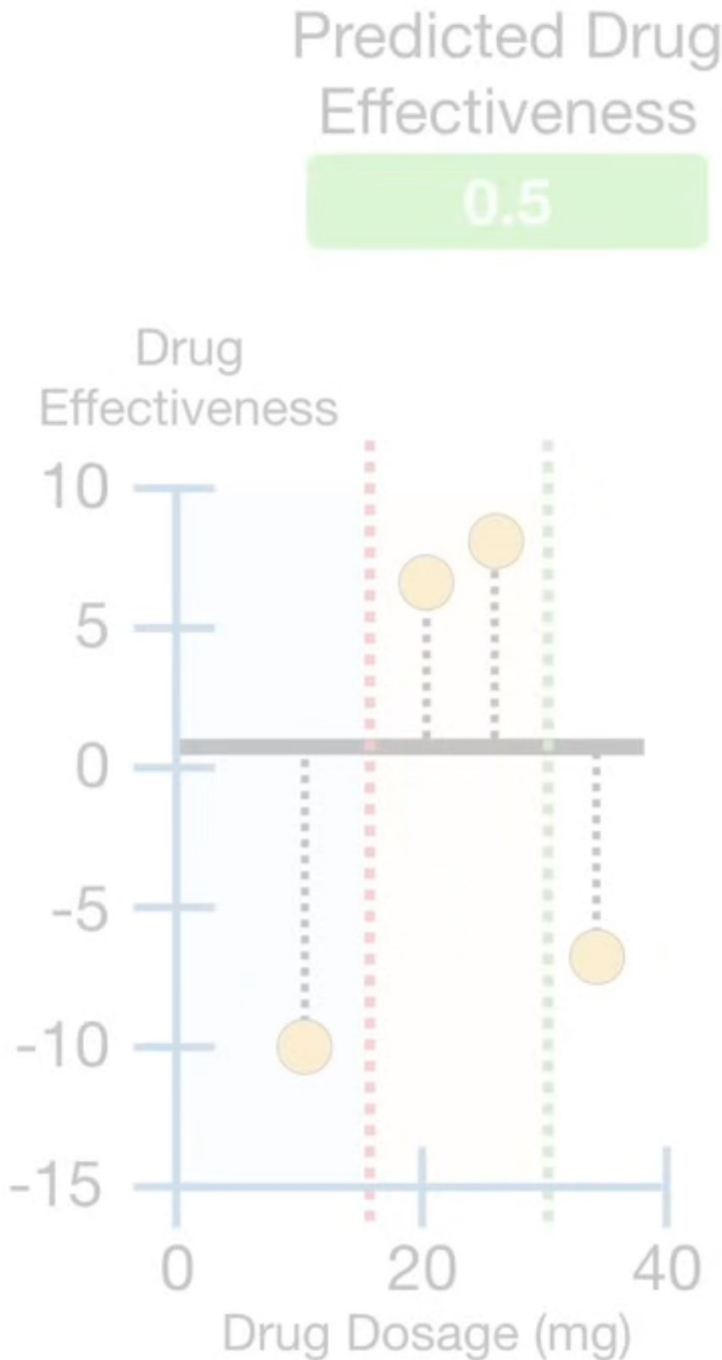
We fit a threshold (gamma) that will be compared to the gain!



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

TERMINOLOGY ALERT!!!
XGBoost calls this number ***γ* (gamma)**.

Kudos to Joshua Starmer!



Gain = 120.33

Dosage < 15

Gain = 140.17

-10.5

Dosage < 30

6.5, 7.5

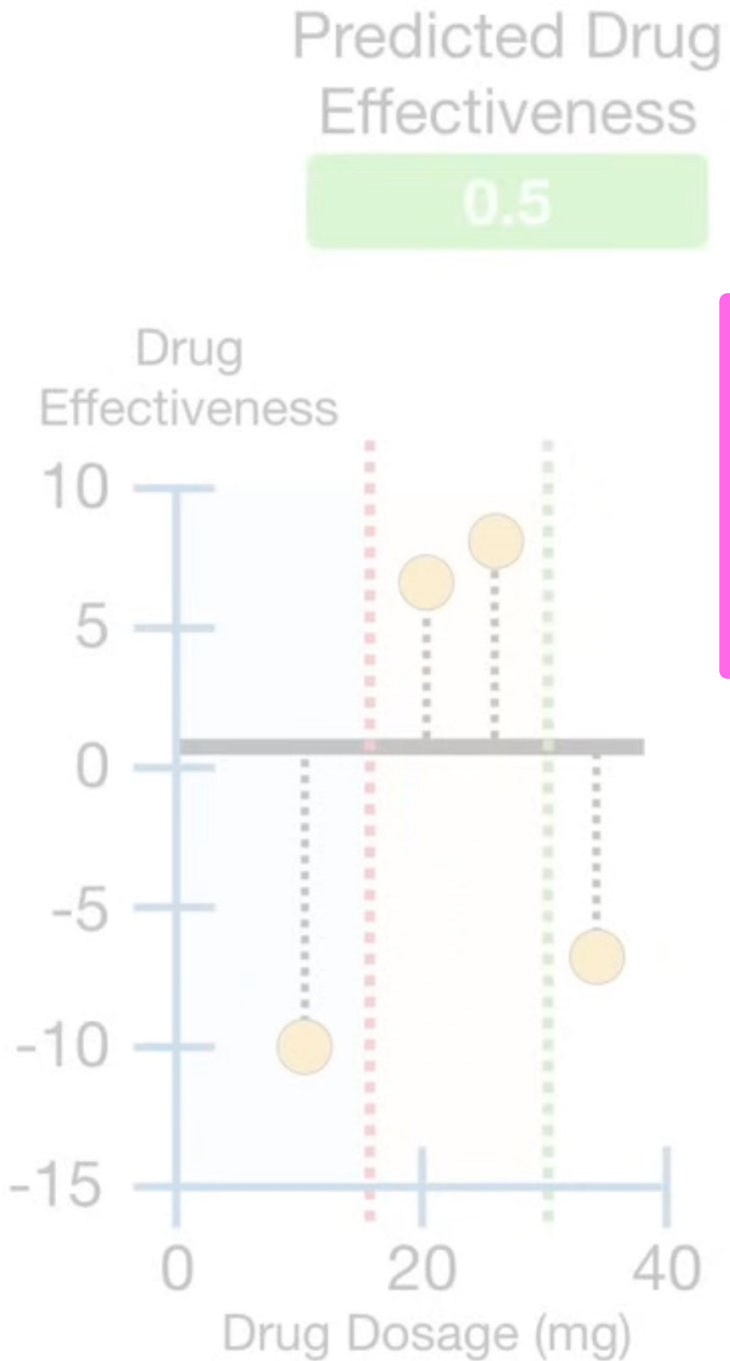
-7.5

Gain → γ (gamma)

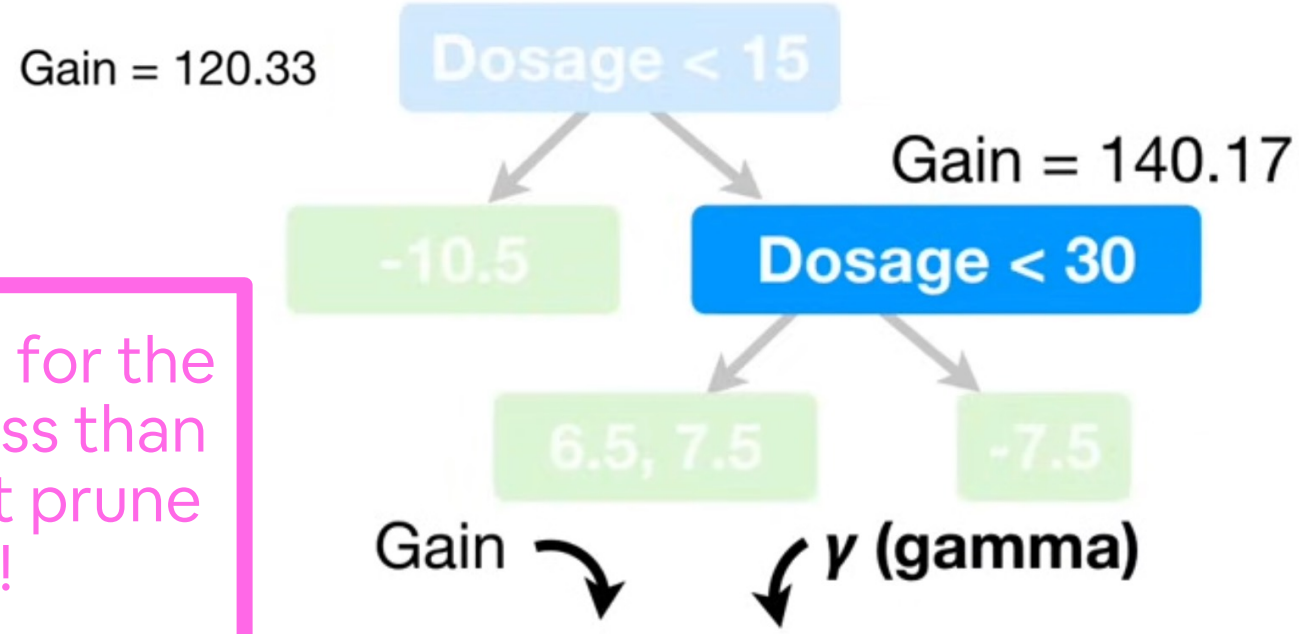
140.17 - 130 = 10.17

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

Kudos to Joshua Starmer!

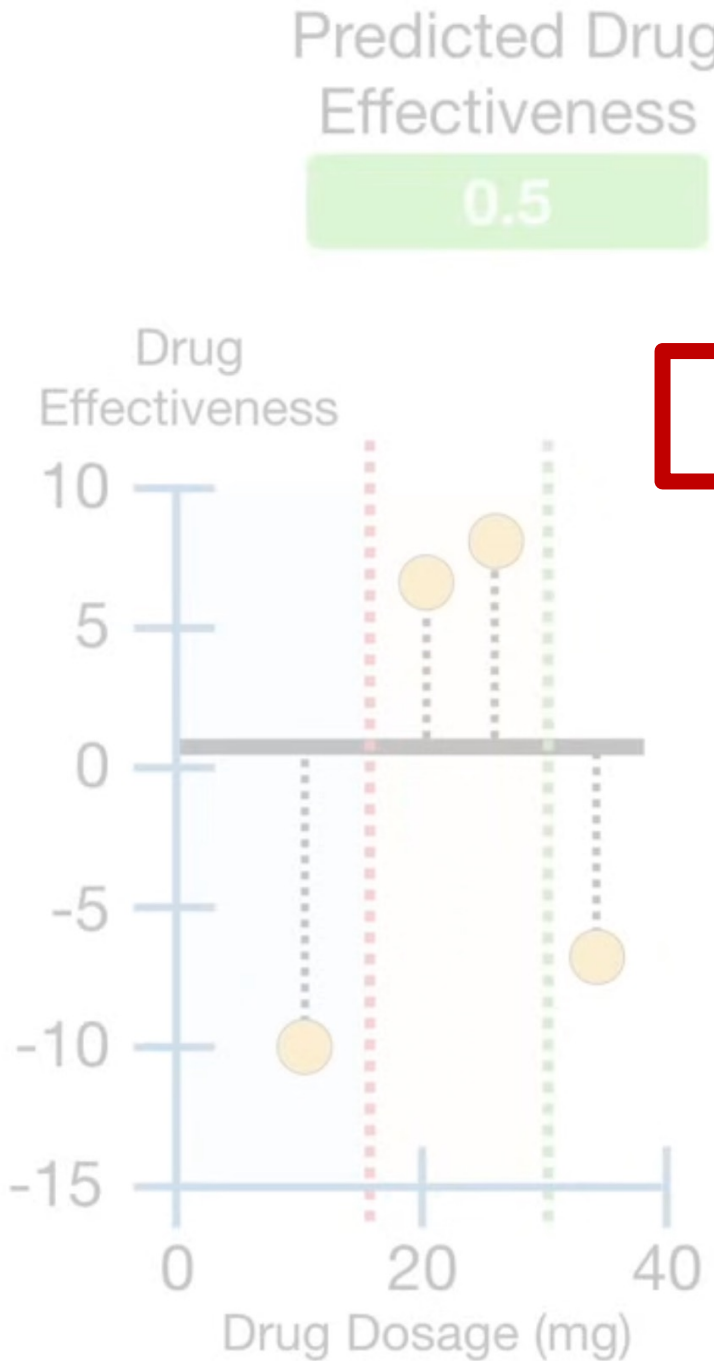


Even if the gain for the root is 120.3 (less than 130) we cannot prune the root!

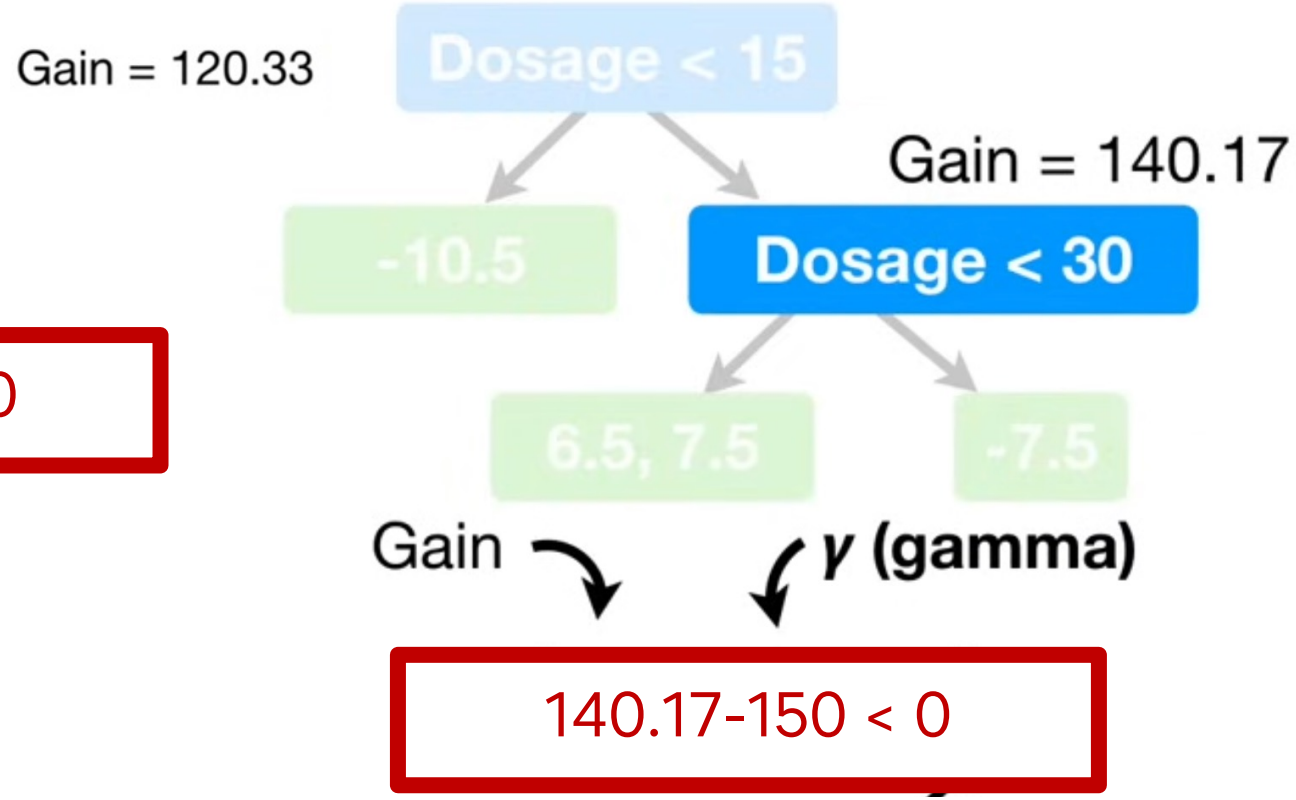


…we get a ***positive*** number, so we will not remove this branch and we are done pruning.

Kudos to Joshua Starmer!



120.33-150 < 0



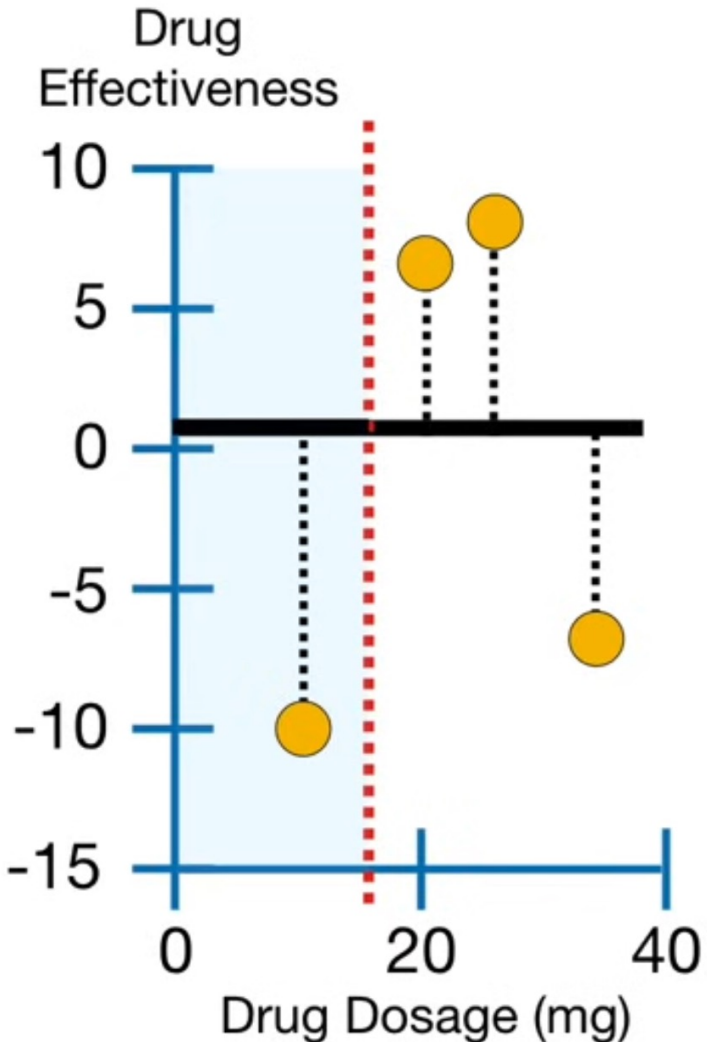
140.17-150 < 0

In this case the increased threshold make us prune the whole tree!

Kudos to Joshua Starmer!

Predicted Drug Effectiveness

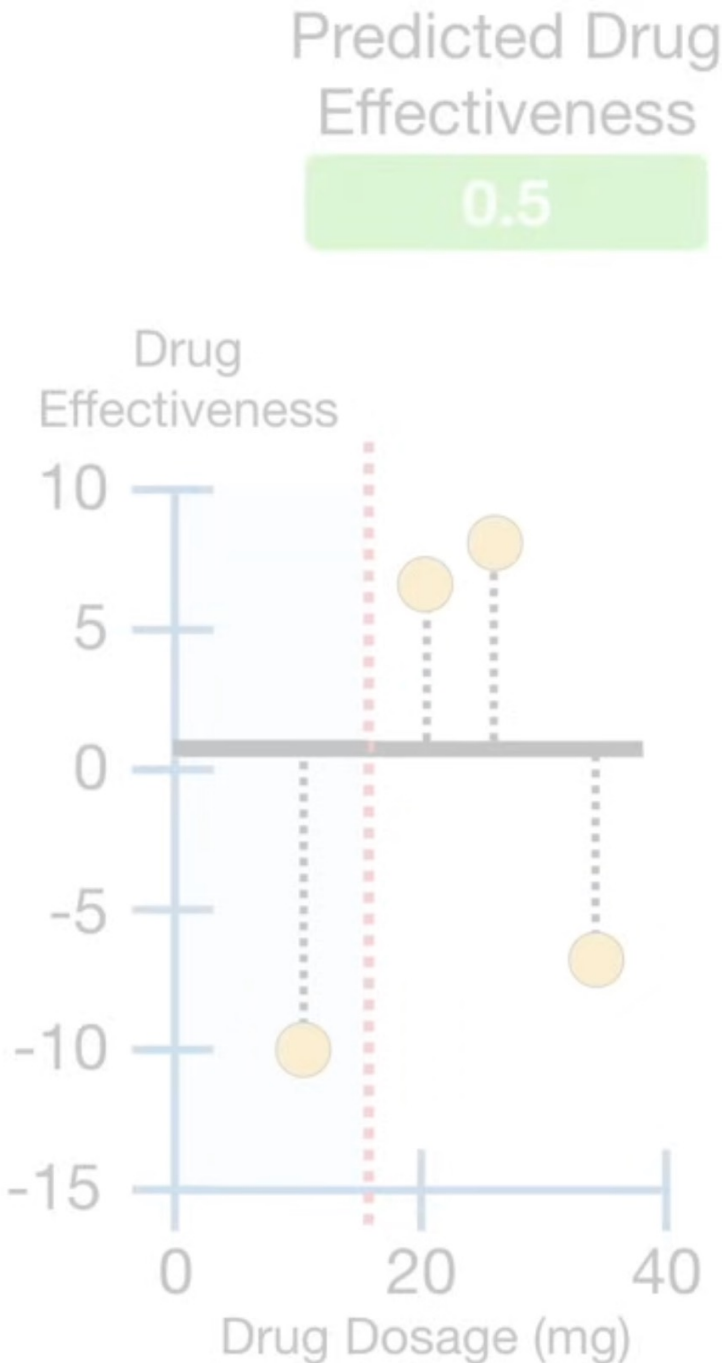
0.5

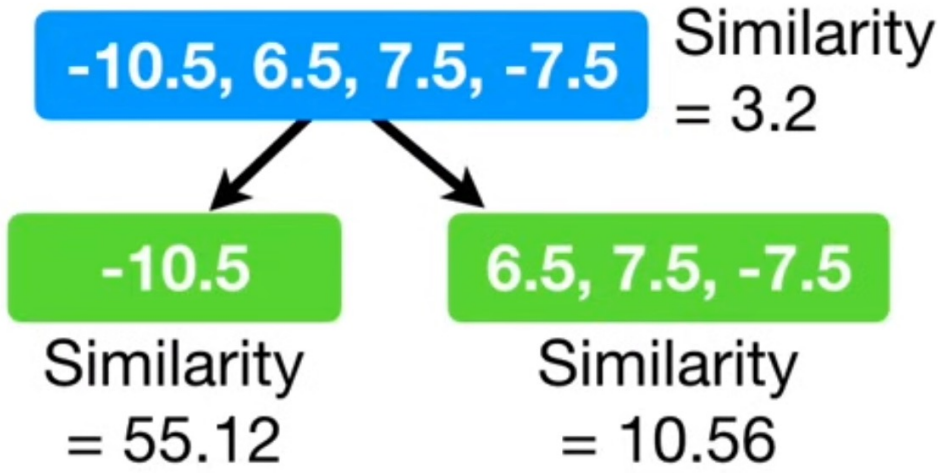


$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + \mathbf{1}}$$

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

Kudos to Joshua Starmer!





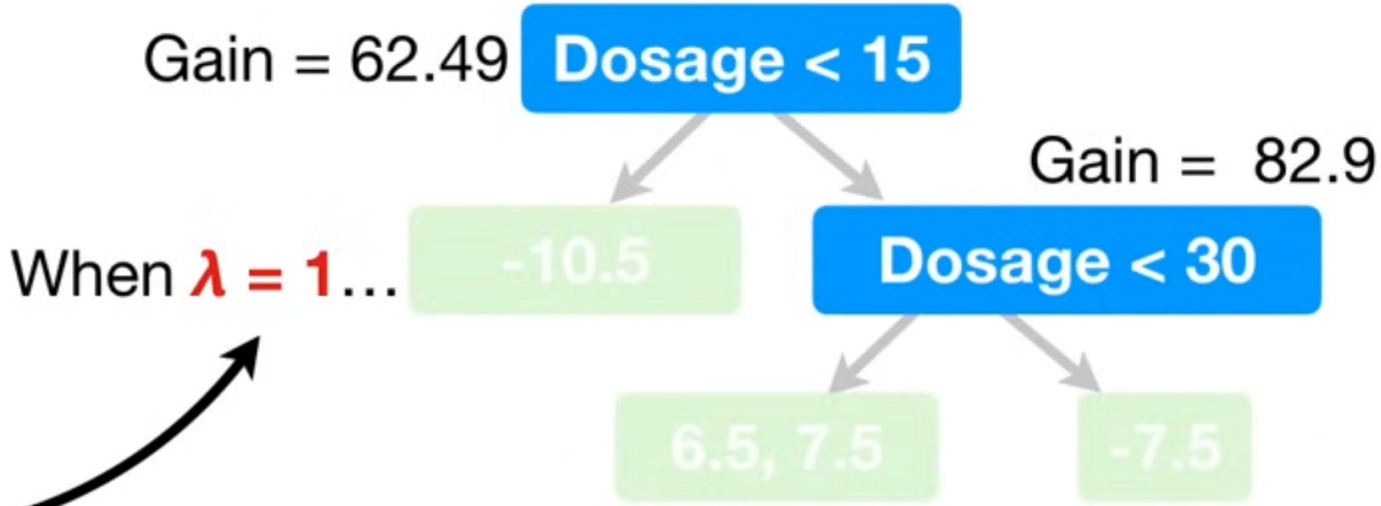
So, one thing we see is that when $\lambda > 0$, the **Similarity Scores** are smaller…

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

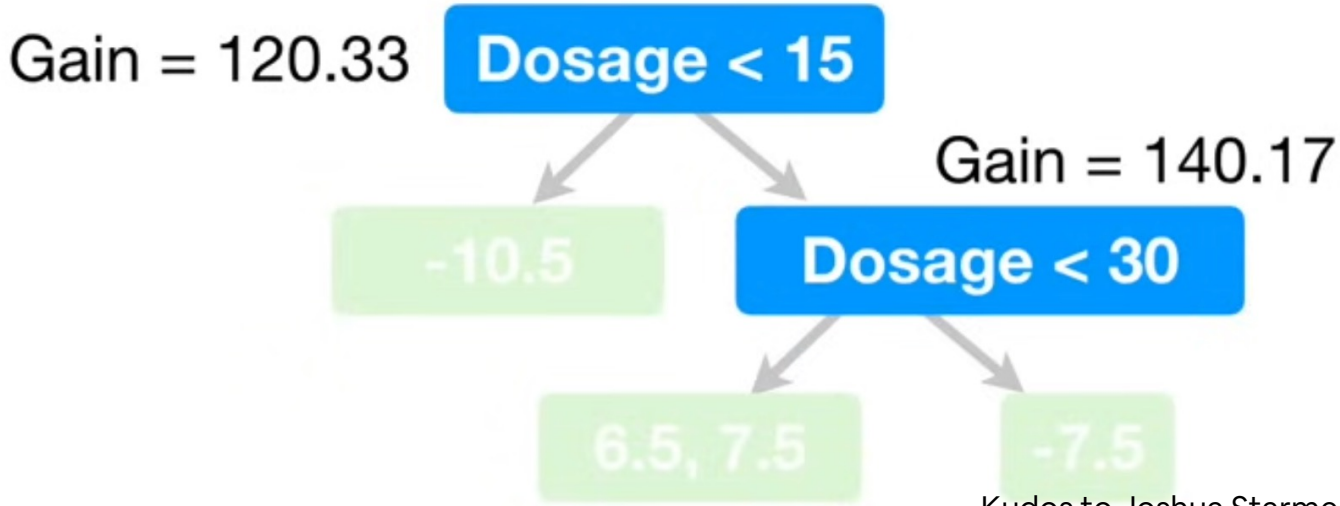
Kudos to Joshua Starmer!

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

When $\lambda = 1$...



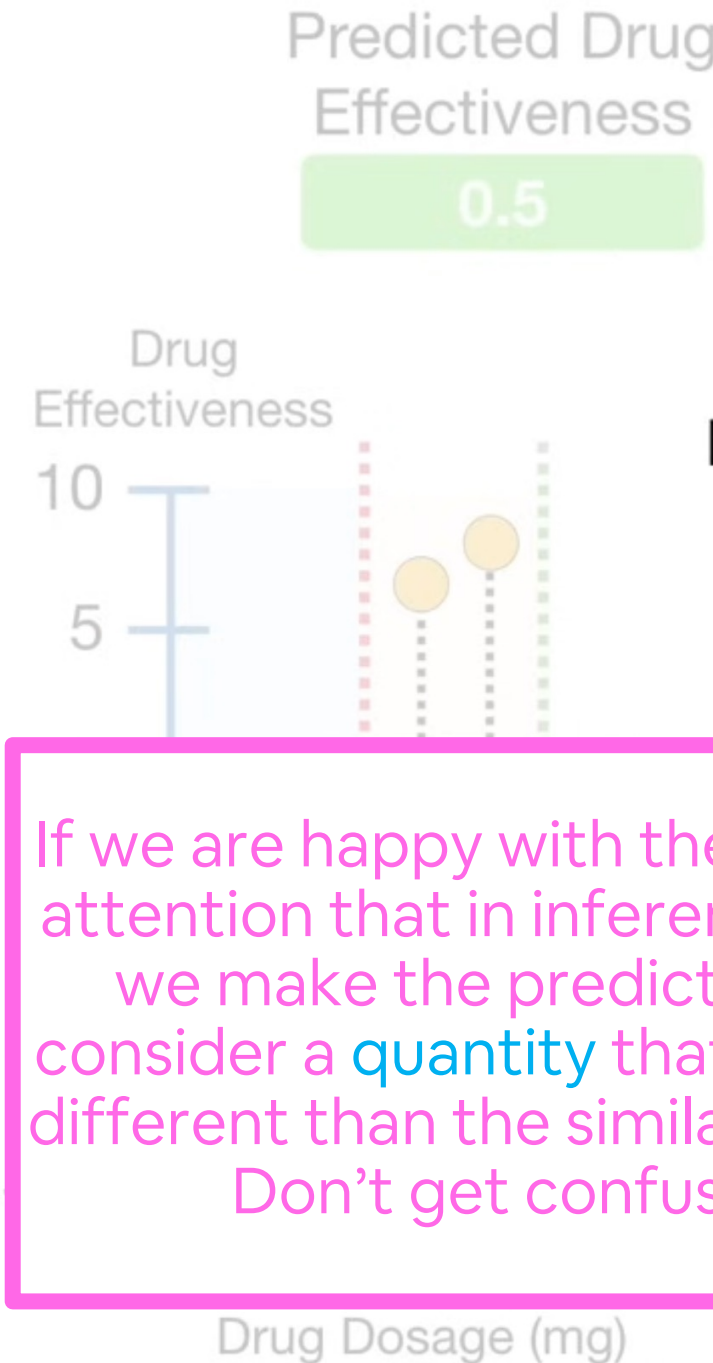
When $\lambda = 0$...

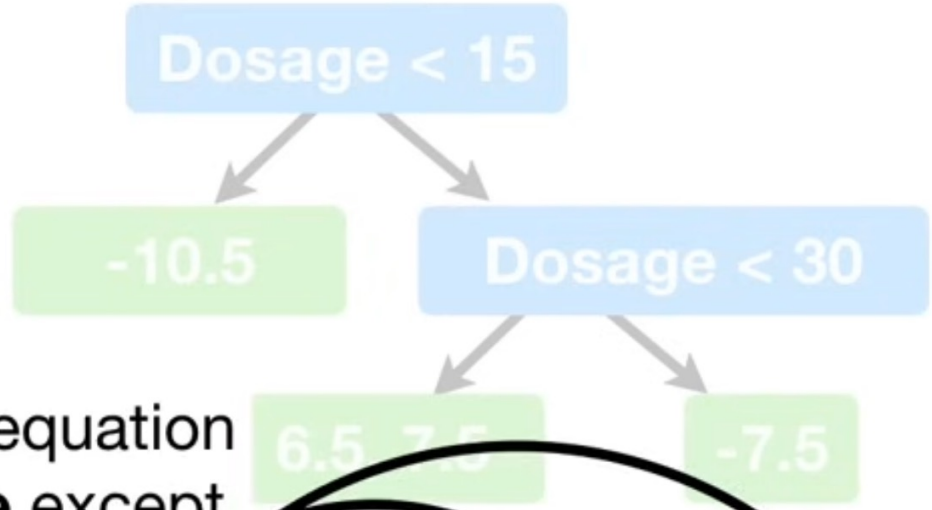


The threshold 130 for example will have a completely different impact!

It is an hyperparameter that can allow us to prevent over fitting the Training data!

Kudos to Joshua Starmer!





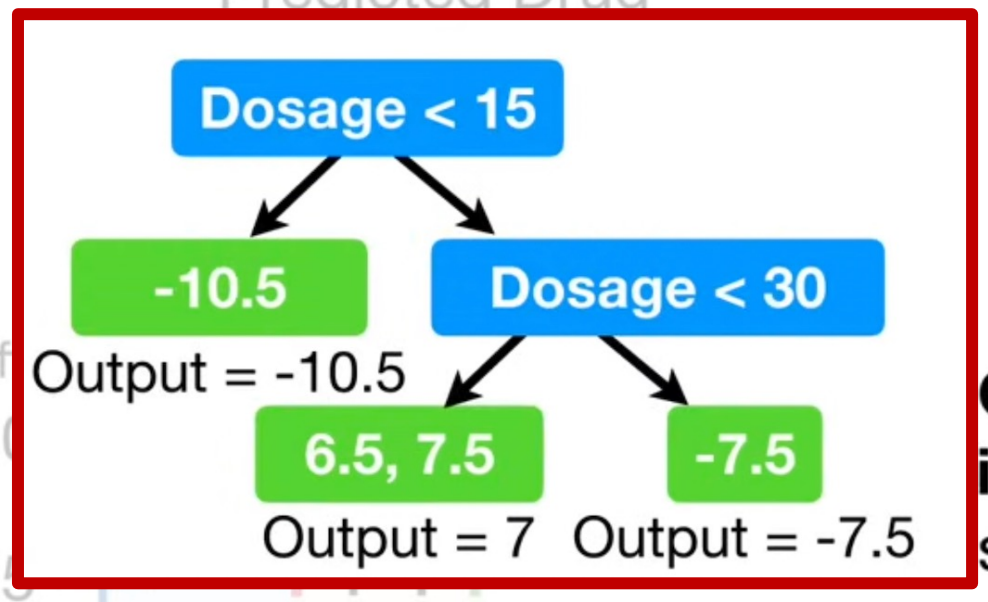
NOTE: The **Output Value** equation is like the **Similarity Score** except we do not square the sum of the residuals.

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

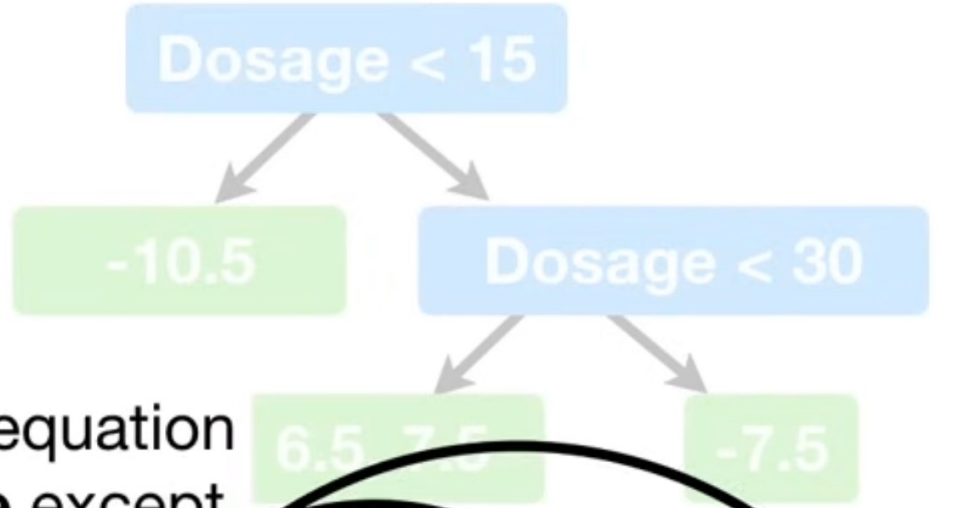
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

If we are happy with the tree, pay attention that in inference (when we make the prediction) we consider a quantity that is slightly different than the similarity score! Don't get confused!

Kudos to Joshua Starmer!



If we are happy with the tree, pay attention that in inference (when we make the prediction) we consider a quantity that is slightly different than the similarity score! Don't get confused!



Output Value equation
imilarity Score except
square the sum of the
residuals.

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

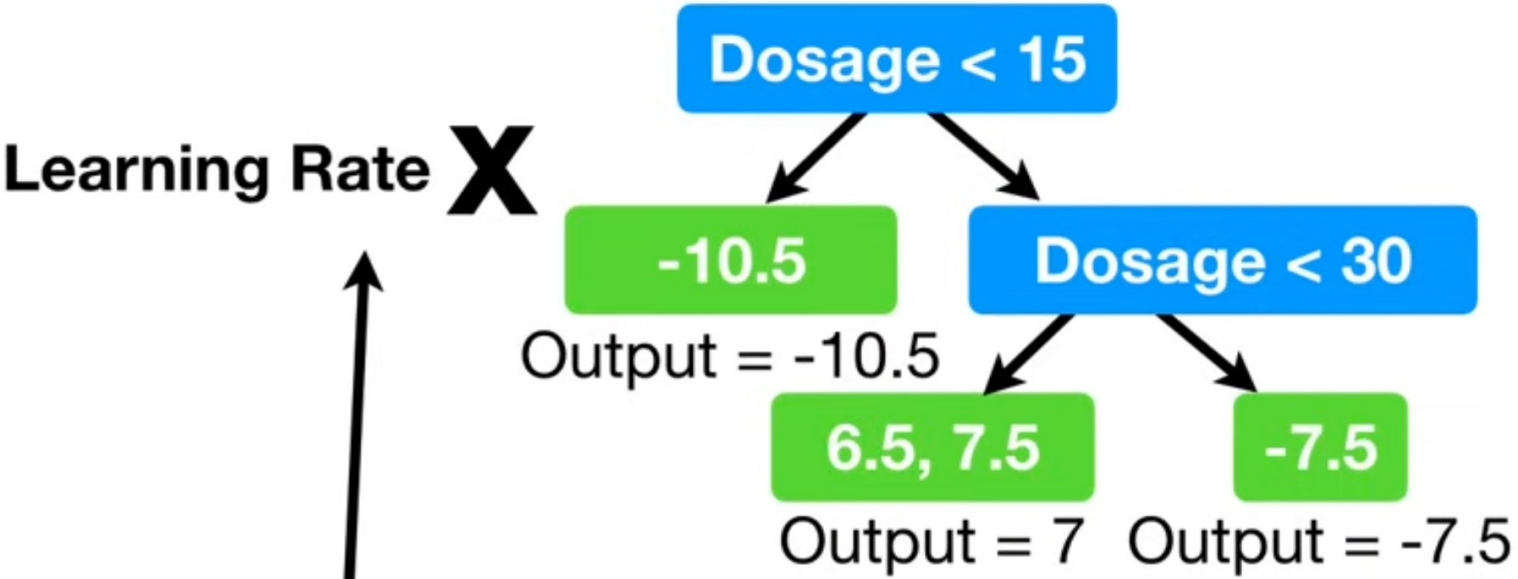
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

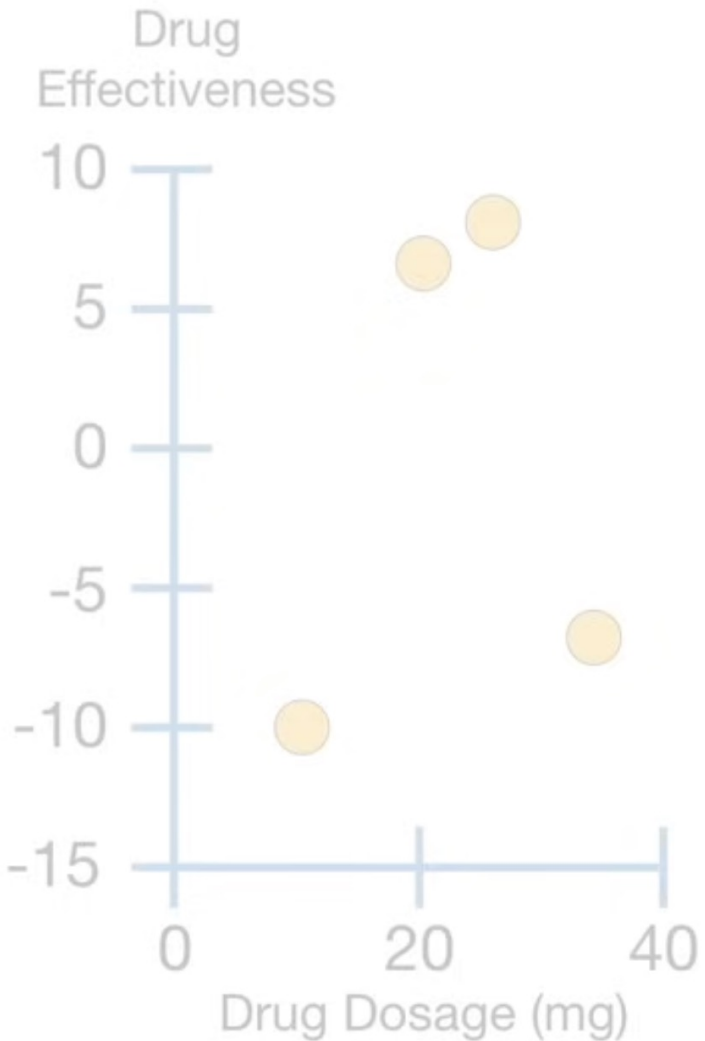
Kudos to Joshua Starmer!

Predicted Drug Effectiveness

0.5

+ Learning Rate X





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

Let's do boosting now (similar structure as gradient boosting)!

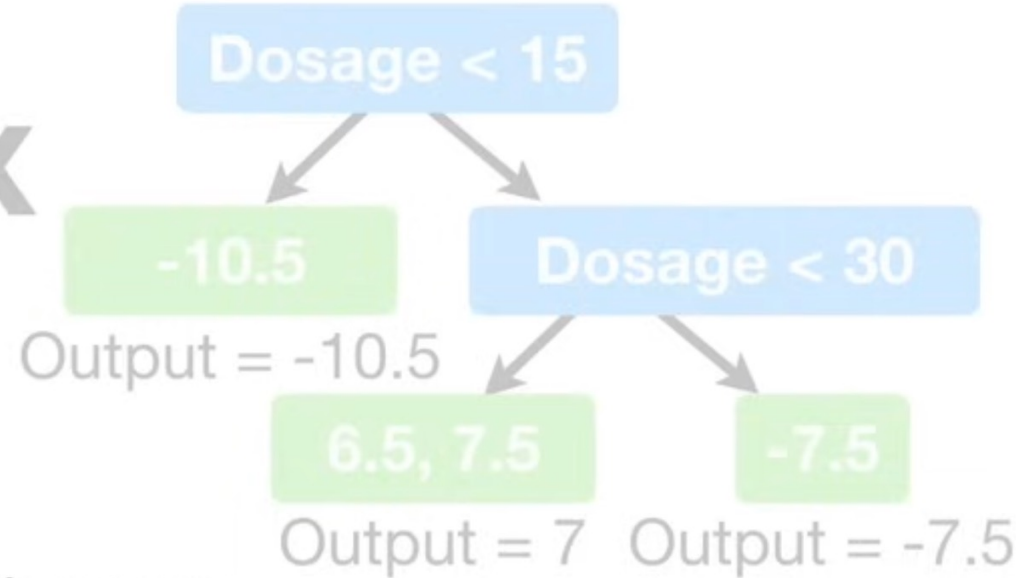
Kudos to Joshua Starmer!

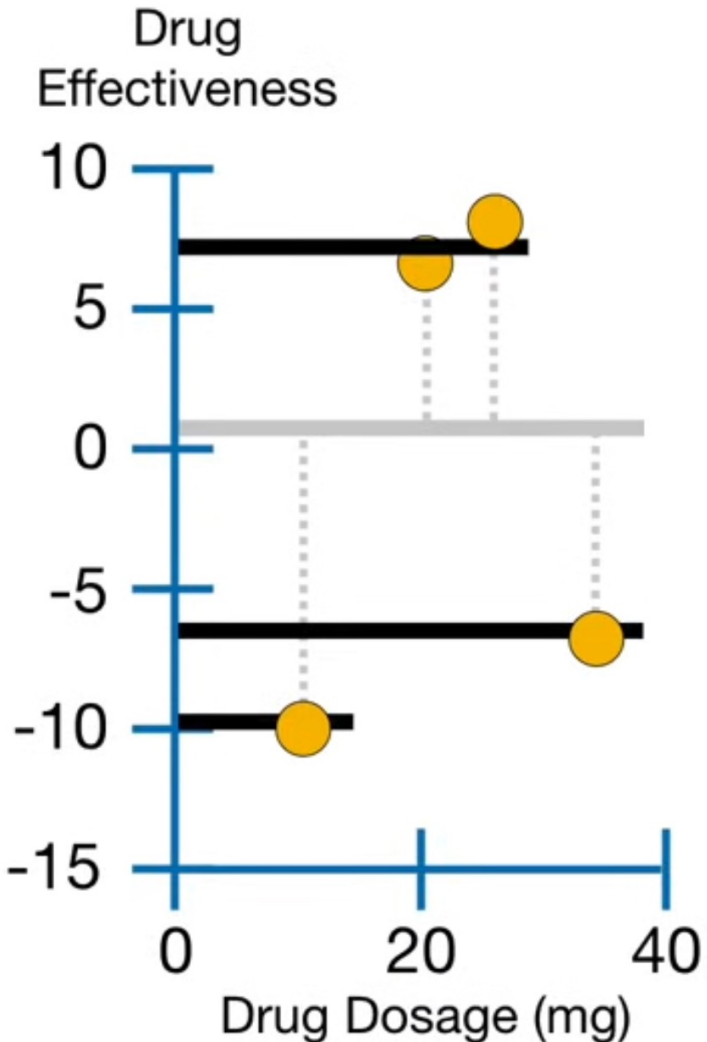
Predicted Drug Effectiveness

0.5

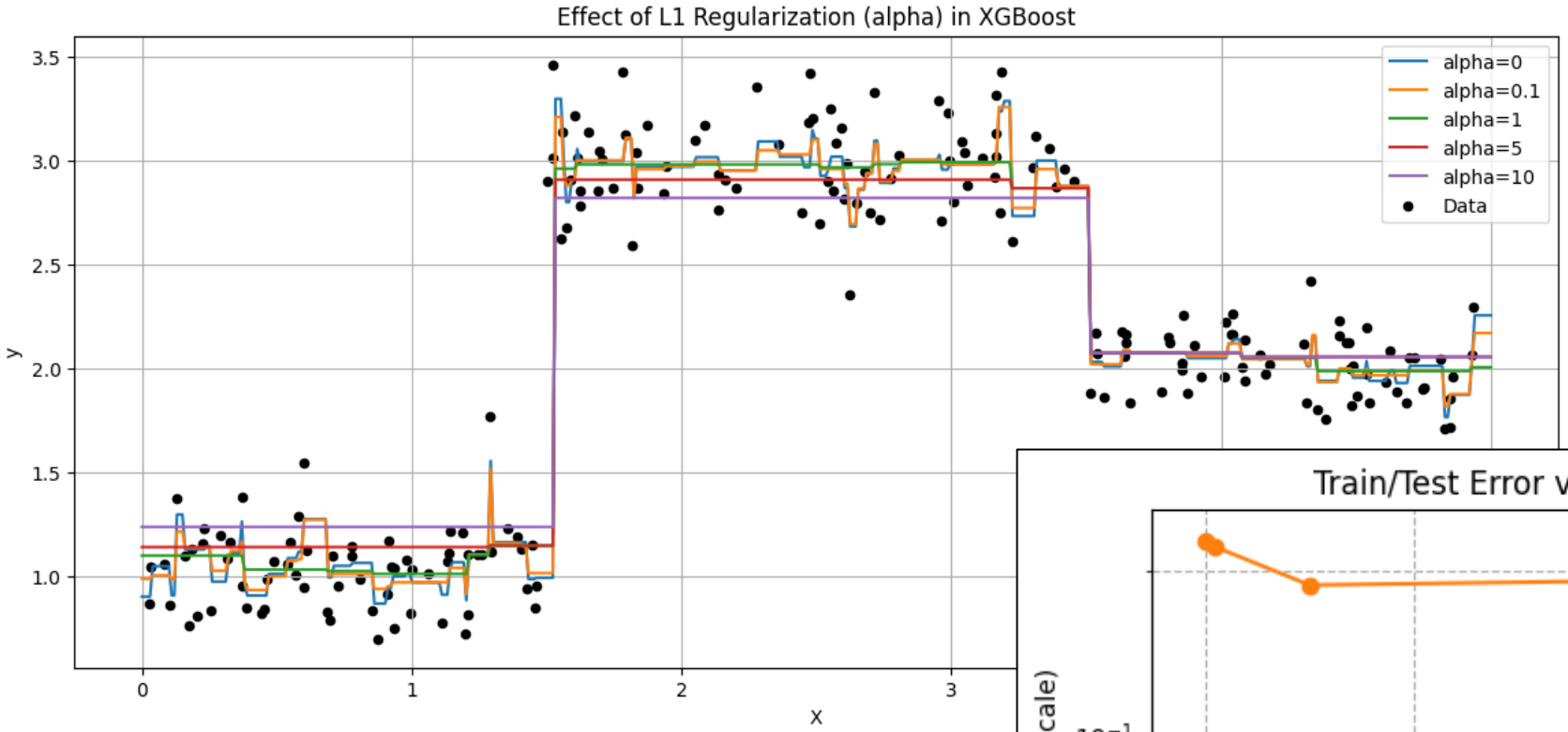
+

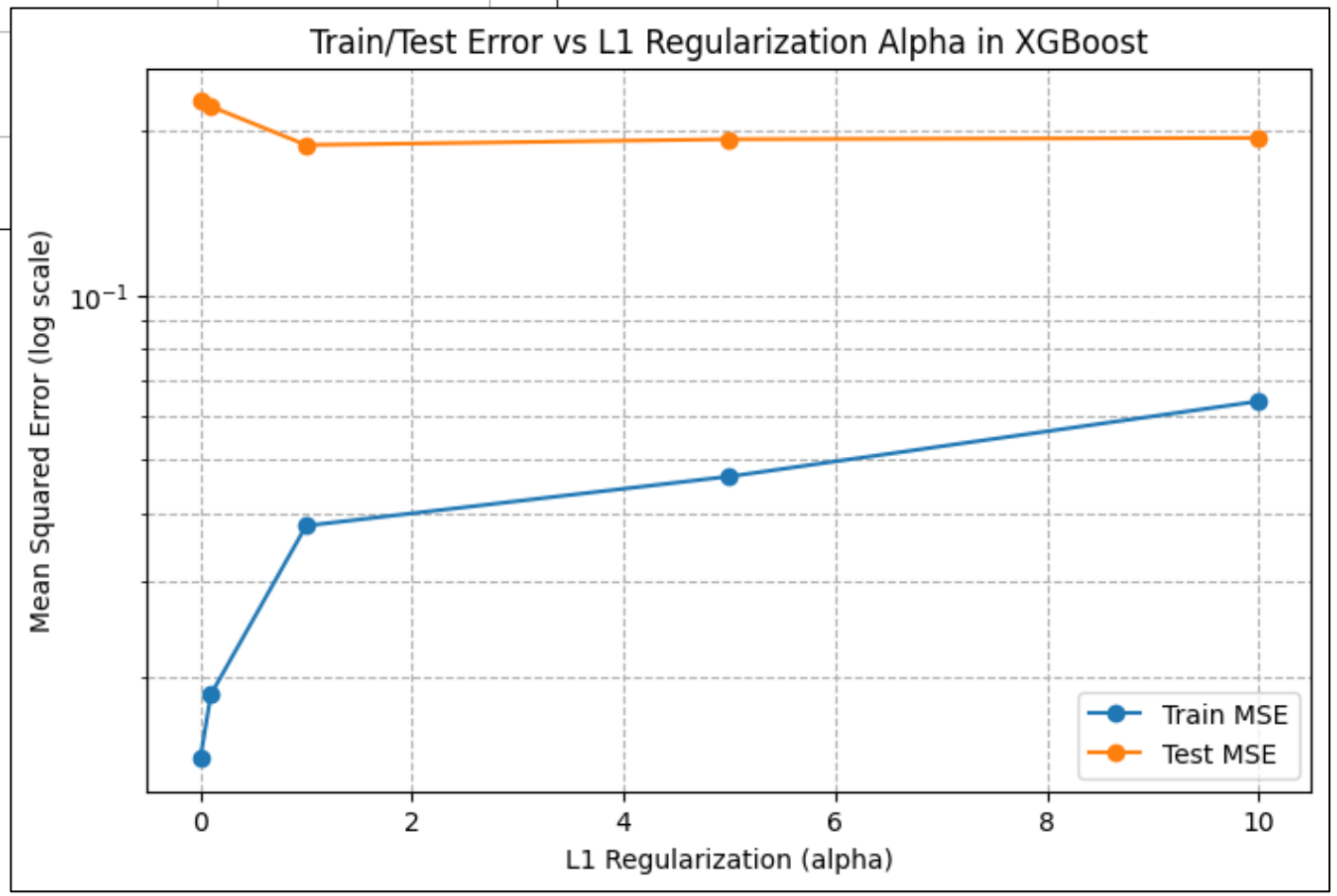
0.3 X





…and we keep building trees until the **Residuals** are super small, or we have reached the maximum number.


Effect of L1 Regularization (alpha) in XGBoost
alpha=0
alpha=0.1
alpha=1
alpha=5
alpha=10
Data
X
y


Train/Test Error vs L1 Regularization Alpha in XGBoost
Mean Squared Error (log scale)
L1 Regularization (alpha)
Train MSE
Test MSE

Method	Core Idea	Model Combination	Key Traits
Bagging	Train trees on random data subsets	Averaging / Voting	Reduces variance, parallelizable, robust to overfitting
Random Forest	Bagging + random feature selection	Averaging / Voting	Strong baseline, good generalization
Boosting	Sequential models to fix previous errors	Weighted sum	Reduces bias, sensitive to noise
AdaBoost	Focus on misclassified samples via reweighting	Weighted sum	Simple, uses weak learners (e.g., stumps), effective on clean data
Gradient Boosting	Fit to loss function gradients	Weighted sum	Flexible loss functions, can overfit without tuning
XGBoost	Regularized GBM with pruning and optimizations	Weighted sum	Fast, regularized, handles missing values
LightGBM	Histogram-based GBM, leaf-wise growth	Weighted sum	Very fast, memory-efficient, great for large-scale problems
CatBoost	Categorical-feature-friendly GBM	Weighted sum	Handles categoricals natively, avoids overfitting
Stacked Ensemble	Combine diverse models with meta-learner	Meta-model (e.g., regression)	Very flexible, risk of overfitting without proper cross-validation

Thank you!

Gian Antonio Susto

Decision Tree (DT): how do we build one?

We will use a ‘recursive’ procedure:

- We start building a tree from the root
- We choose the variable to be associated to the decision based on the one that better ‘simplifies’* the problem: a scenario where we have a dominant/only class
- We iterate this, until classes are separated or until we reach a given ‘depth’ of the tree

*We need a quantitative metric to define how ‘simple’ a decision is at a leaf level!