Bagging and Boosting in R

BRUG Meetup
February/2020 series
Bangalore, India

Kamal Mishra | Mohammad J Asad

23-Feb-2020

Disclaimer

The content and/or postings here are personal point of views from our experiences, thoughts, readings from various sources and don't necessarily represent any firm's positions, strategies and/or opinions.

Agenda

- Background
- Examples around motivation
- Need for the usage
- Difference between Bagging and Boosting
- Algorithms
- Deep dive on few techniques
- Different practice usage in R
- References
- Q&A

Background

- Single decision tree does not perform well. Very simple, of course easy to interprete, however underperforms. Does not work for complex business scenarios
- Variance ?? – reduction !!
- Aggregating multiple decision trees for same training dataset performs better in test data - >> bagging
- Aggregate multiple decision trees >> reduces variance… and hence improves accuracy and prediction performance

Examples around motivation

- Let's look at sample data for understanding and illustrative comparison
- Car Seats dataset: A simulated data set containing sales of child car seats at 400 different stores. – comes default in R

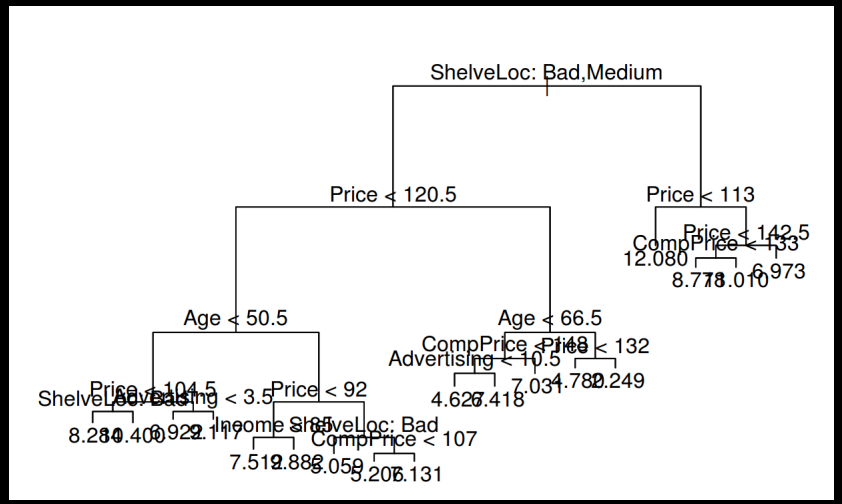
```
> head(Carseats)
  Sales CompPrice Income Advertising Population Price ShelveLoc Age Education Urban  US
1  9.50       138     73          11        276   120       Bad  42        17   Yes Yes
2 11.22       111     48          16        260    83      Good  65        10   Yes Yes
3 10.06       113     35          10        269    80    Medium  59        12   Yes Yes
4  7.40       117    100           4        466    97    Medium  55        14   Yes Yes
5  4.15       141     64           3        340   128       Bad  38        13   Yes  No
6 10.81       124    113          13        501    72       Bad  78        16    No Yes
```

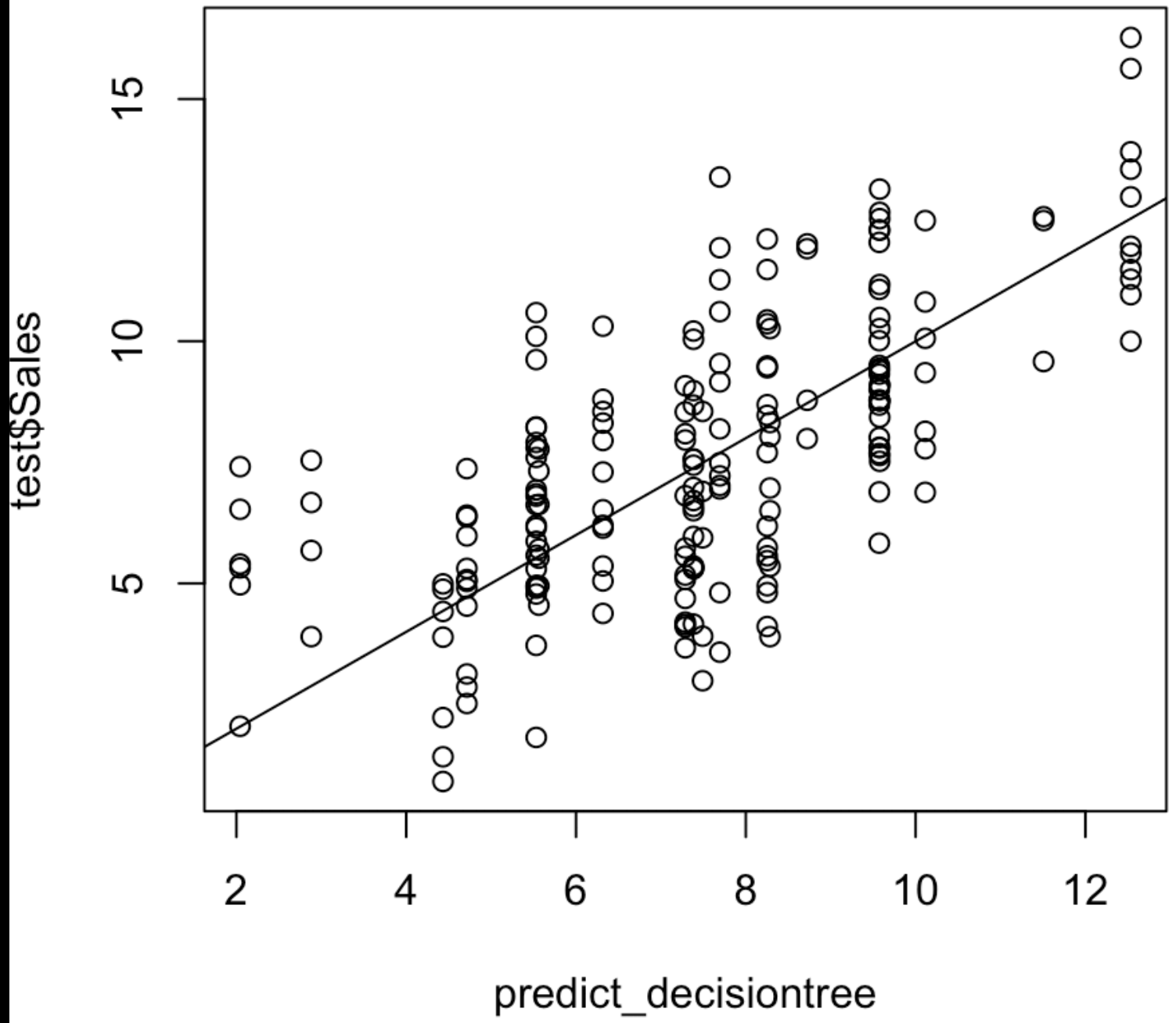
```
> str(Carseats)
'data.frame':   400 obs. of  11 variables:
 $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
 $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
 $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
 $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
 $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
 $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
 $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
 $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
 $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

Decision Tree

- MSE = 4.74

```
> model_decisiontree = tree(Sales ~ .,data = train)
> predict_decisiontree = predict(model_decisiontree,test)
> mean((predict_decisiontree-test$Sales)^2)
[1] 4.7397
```



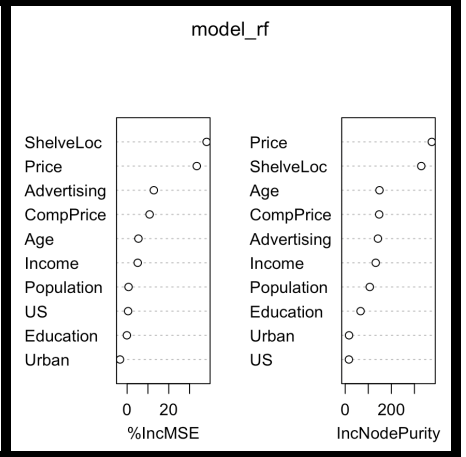


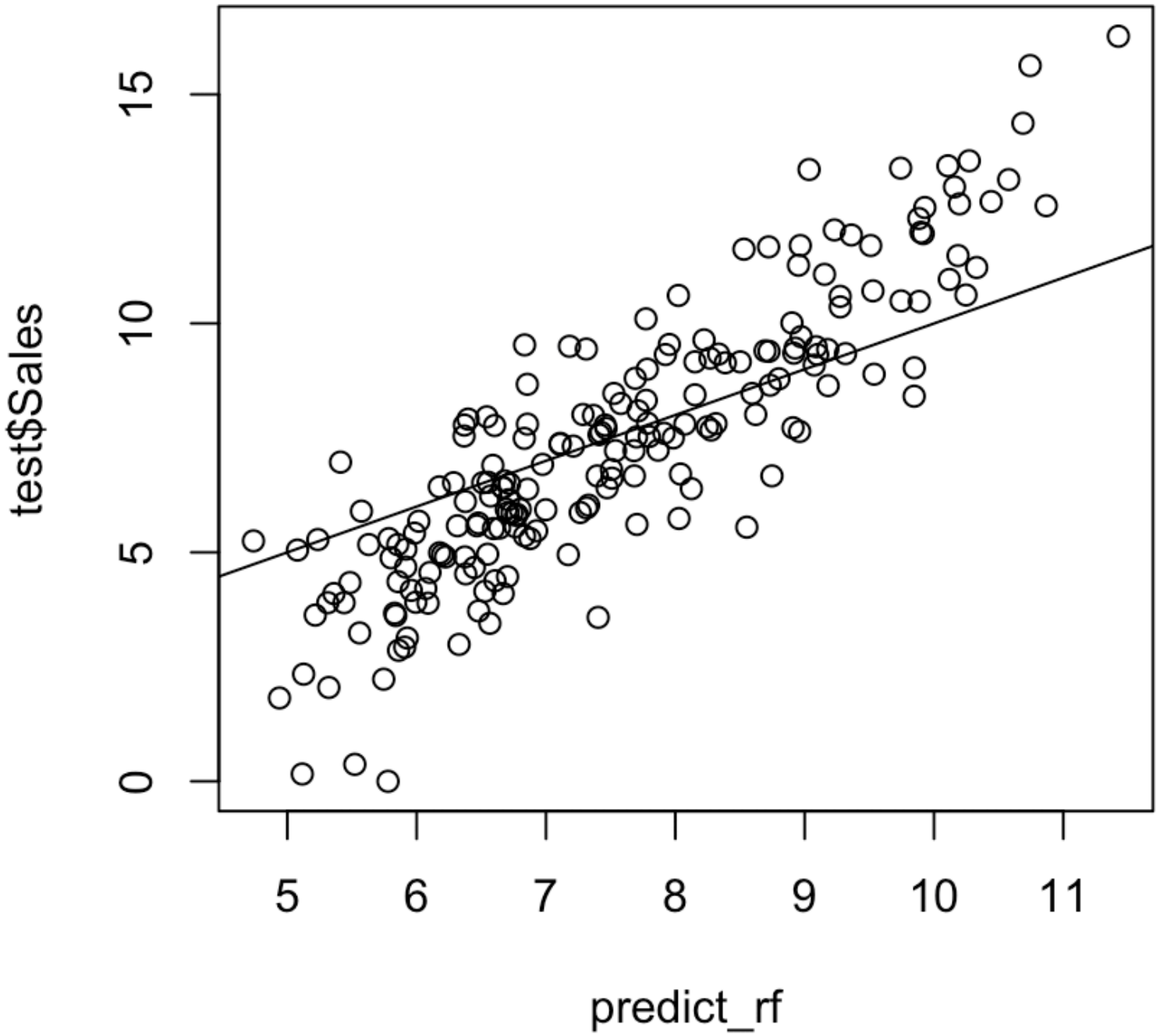
Random Forest

- MSE = 3.10

```
> # Random Forest
> set.seed(124)
> train = sample(1:nrow(Carseats), 200)
> test = Carseats[-train,]
> model_rf = randomForest(Sales ~ ., data = Carseats,subset = train,importance=TRUE)
> predict_rf = predict(model_rf, test)
> mean((predict_rf-test$Sales)^2)
[1] 3.10543
```

```
> importance(model_rf)
               %IncMSE IncNodePurity
CompPrice   10.7924666     147.24011
Income       5.0869884     132.31357
Advertising 12.8185917     141.83741
Population   0.6821209     106.47369
Price       33.4472789     375.15465
ShelveLoc   38.1862771     329.59710
Age          5.4278140     148.50072
Education   -0.1189124      66.65301
Urban       -3.4175289      16.74292
US           0.4653299      16.14888
```

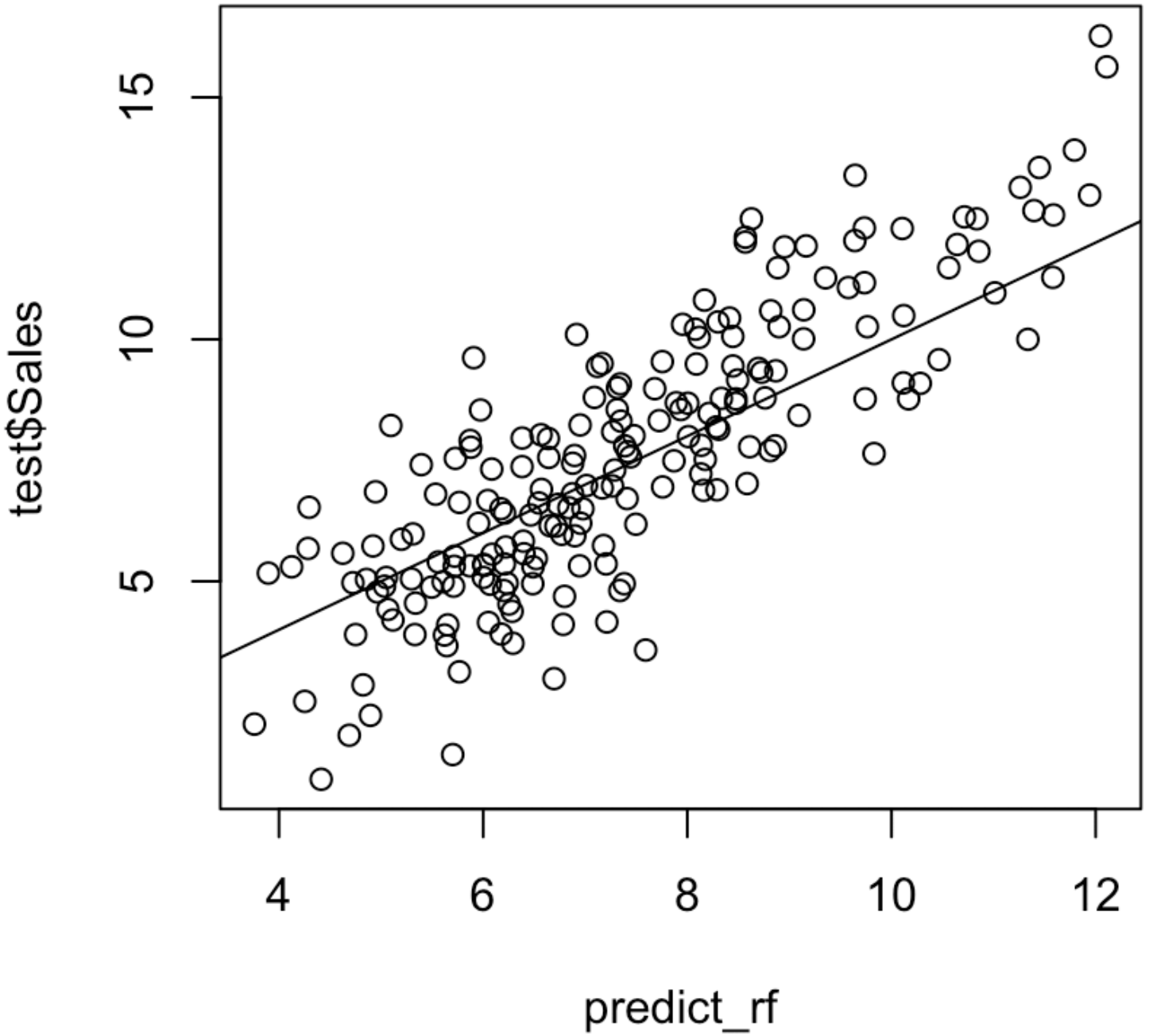




Random Forest
(with specific features)

- MSE = 2.67

```
> set.seed(124)
> model_rf = randomForest(Sales ~ ., data = Carseats,subset = train,mtry=6,importance=TRUE)
> predict_rf = predict(model_rf, test)
> mean((predict_rf-test$Sales)^2)
[1] 2.672282
> plot(predict_rf,test$Sales)
> abline(0,1)
> summary(model_rf)
```

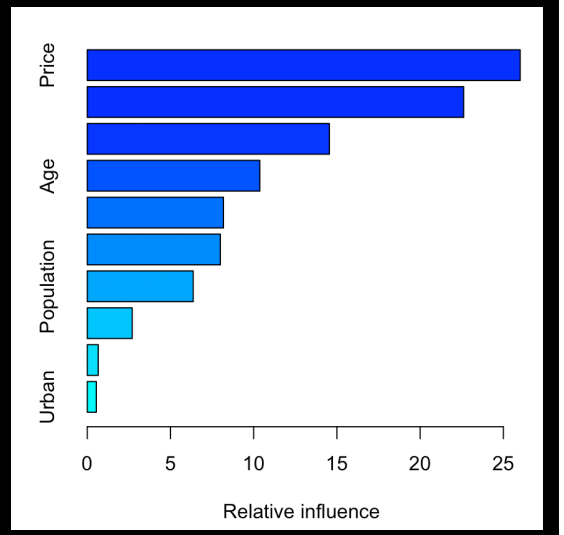


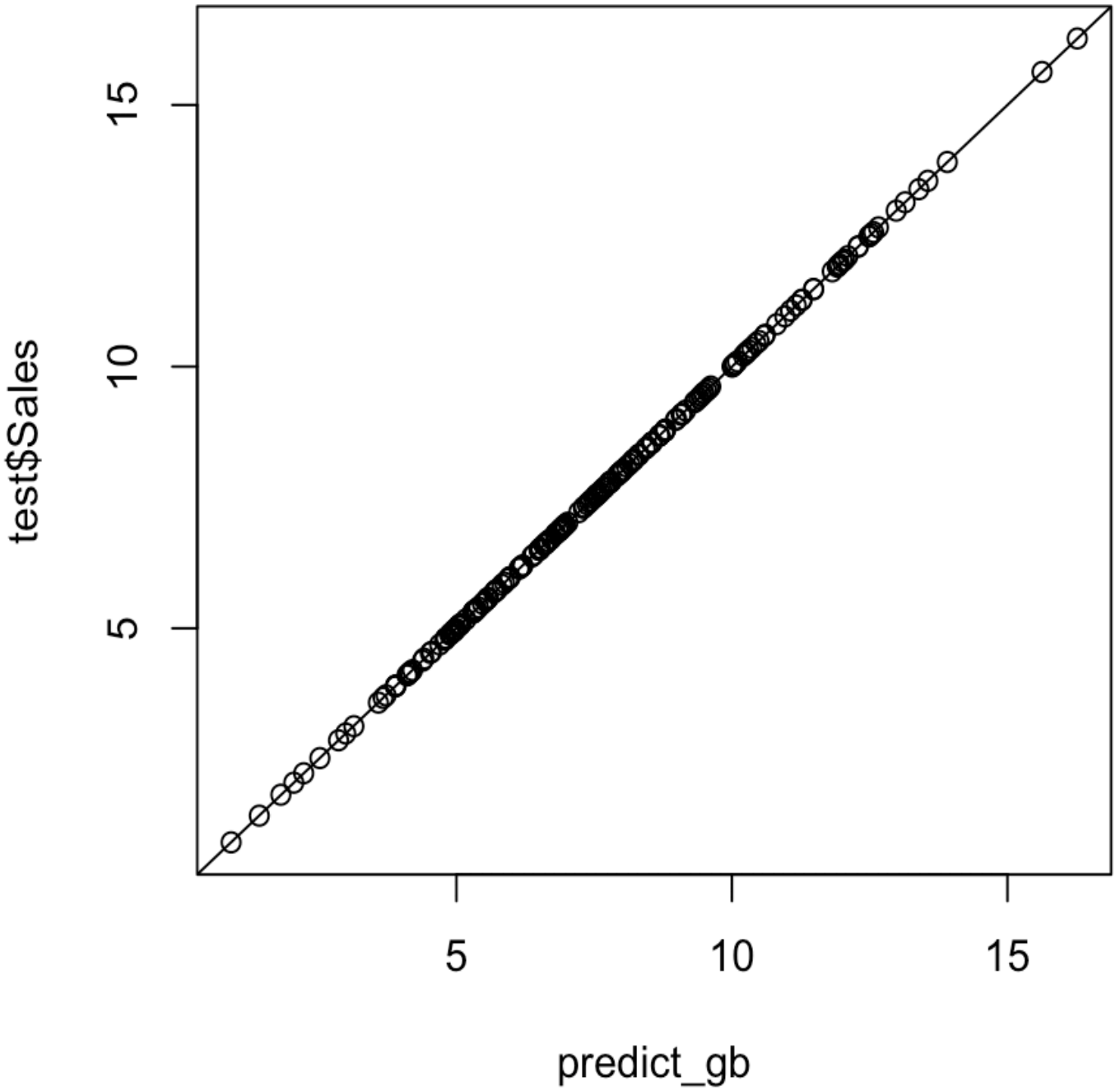
GBM
(Gradient Boosting)

- MSE = 0.00001193749

```
> # GBM
> set.seed(124)
> model_gbm = gbm(Sales ~ ., data = Carseats, distribution = "gaussian",
+                 n.trees = 5000, interaction.depth = 4)
> predict_gb = predict(model_gbm, newdata = test,n.trees = 5000)
> mean((predict_gb-test$Sales)^2)
[1] 1.193749e-05
```

```
> summary(model_gbm)
                       var    rel.inf
Price                Price 26.0106153
ShelveLoc        ShelveLoc 22.6164153
CompPrice        CompPrice 14.5485276
Age                    Age 10.3700471
Advertising Advertising    8.1831494
Income              Income  7.9976652
Population      Population  6.3666170
Education        Education  2.7017348
US                      US  0.6602384
Urban                Urban  0.5449897
```





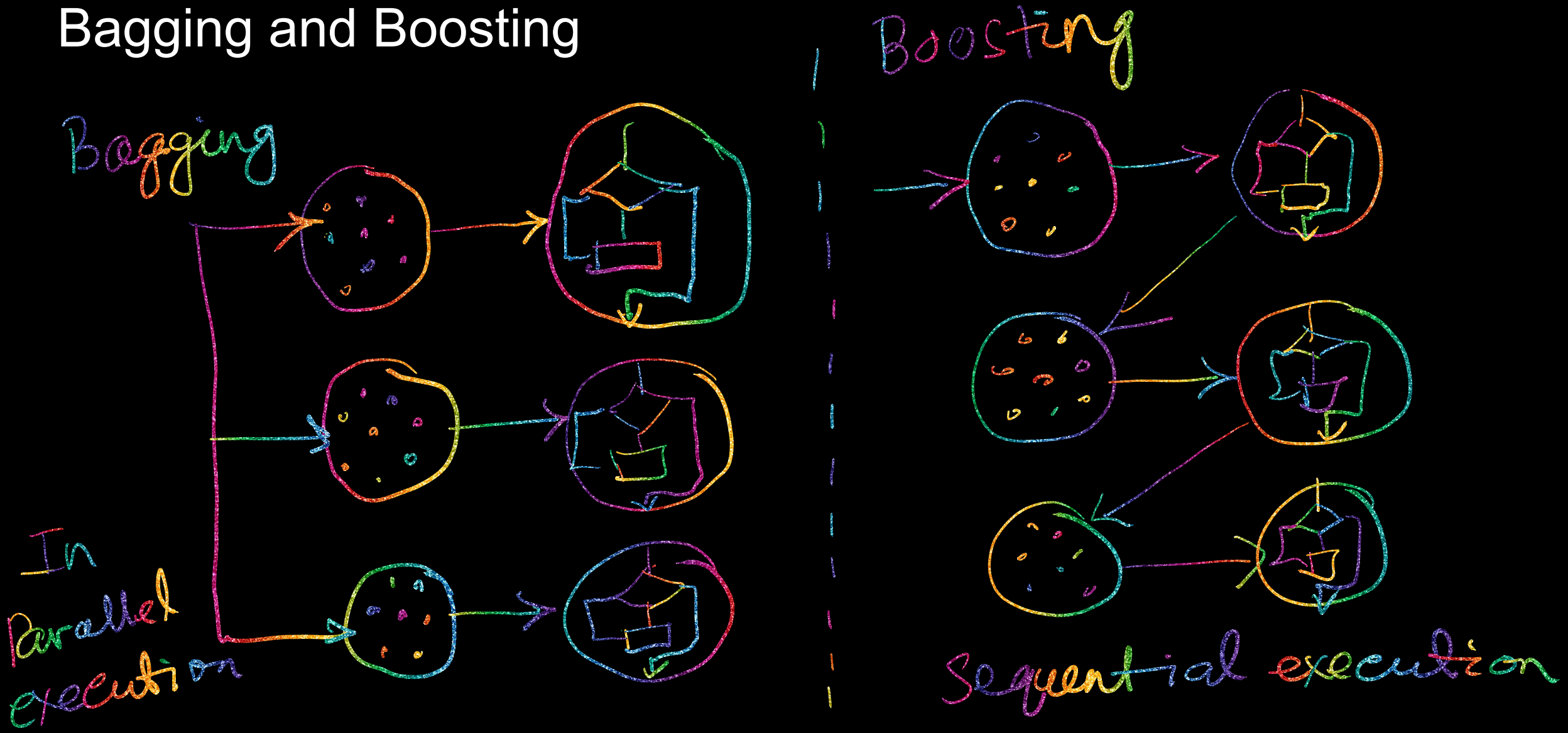
Need for the usage

- Improvise on accuracy, predictive performance
- Ensemble process to increase accuracy and reduce variation
- In case of Bagging: Important feature is around – variable importance
- Can not visualize in bagging scenario (Accuracy goes up, Interpretability down) compared to the way it used to be in simple decision trees
- Ranking of features happen based on different type of techniques:
  - For regression – Residual Sum of squares
  - For classification – Gini coefficient
- Different tactics for bagging and boosting
  - Bagging – parallel execution philosophy
  - Boosting – sequential execution philosophy

Bagging vs Boosting

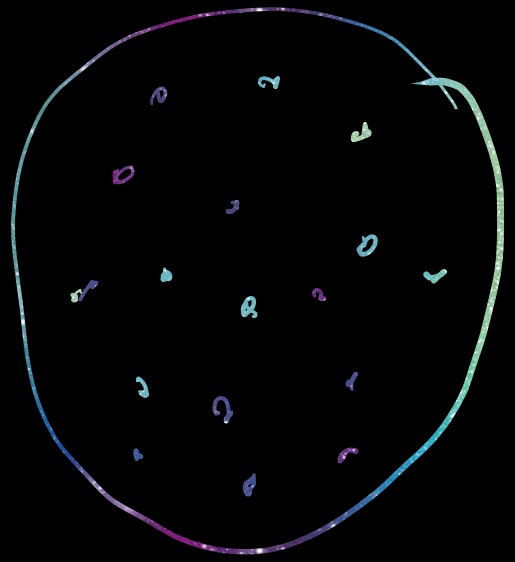
Similarities	Dis-similarities
○ Both are ensemble methods ○ Use random sampling to generate several training sets ○ Make the final decision by averaging N learners or by majority voting (depending on Regression / Classification type of problems) ○ Good at reducing variance ○ Provide higher stability and better predicting power for accuracy	○ Bagging uses parallel execution whereas Boosting performs sequential execution ○ Multiple models built independently for Bagging (N models), whereas Boosting experiments including new models that do well where previous models fail ○ Boosting determines weighted data approach to tip the scales in favor of difficult cases. i.e. it gives more weightage to those that performs well on training data ○ Boosting tries to reduce bias ○ Bagging solves the over-fitting issue while Boosting may even increase it.

Bagging and Boosting

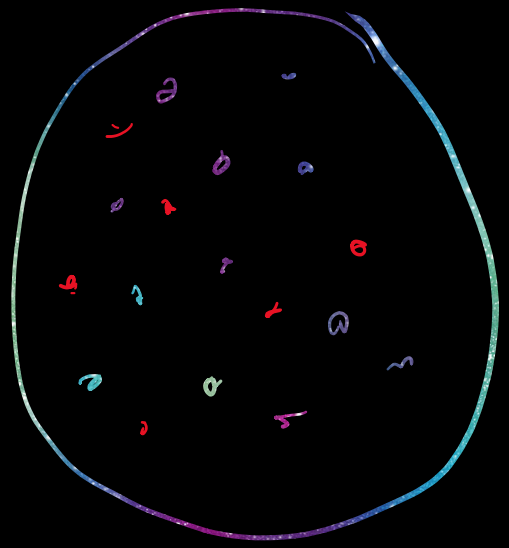


Difference in philosophies illustrated

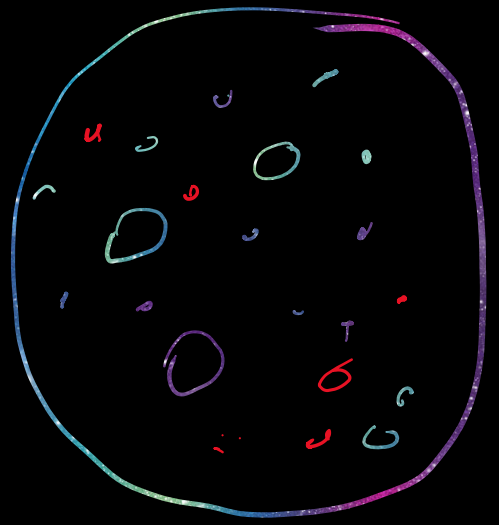
Normal single tree based learning vs Bagging vs Boosting



Complete training set is being used. This is for a single tree based learning approach



Random sampling with replacement is being used. Bootstrapping is used. This is bagging scenario.



Random sampling with replacement over weighted data in a sequential manner. This is boosting scenario.

Algorithms

- Random Forest
- Gradient Boosting (GBM)
- AdaBoost (Adaptive Boosting)
- XGBoost (eXtreme Gradient Boosting)
- catBoost
- LightGBM (Light Gradient Boosting)

Deep dive on few algorithms…

Random Forest

Original Dataset

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

Bootstrapped Dataset

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

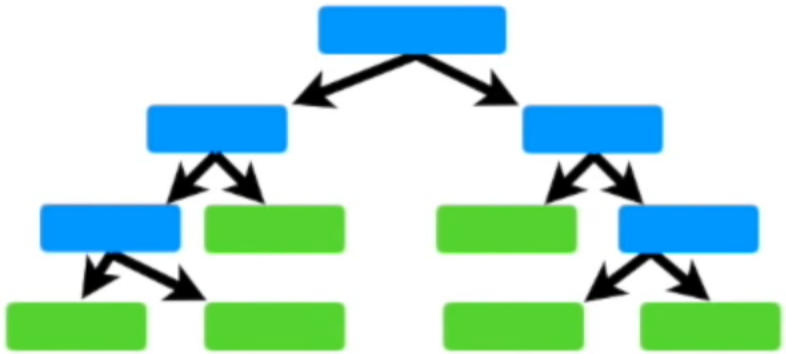
…and here it is.

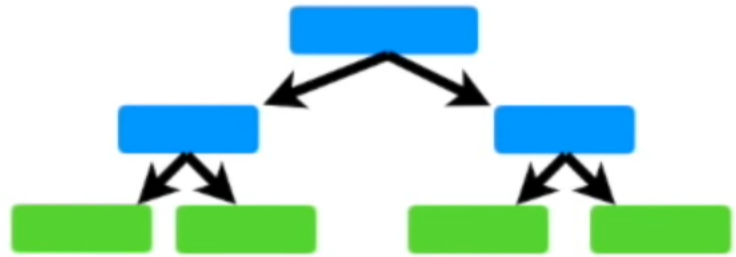
Step 2: Create a decision tree using the bootstrapped dataset, but only use a random subset of variables (or columns) at each step.

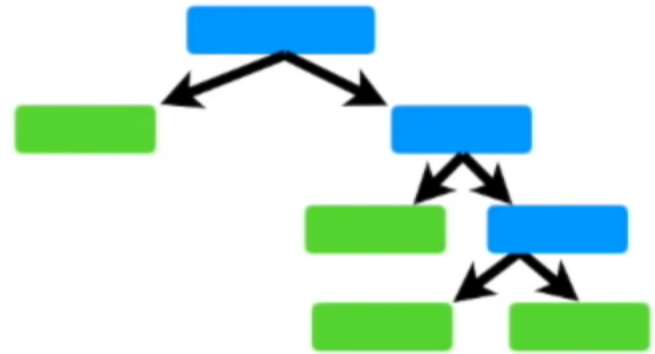
Bootstrapped Dataset

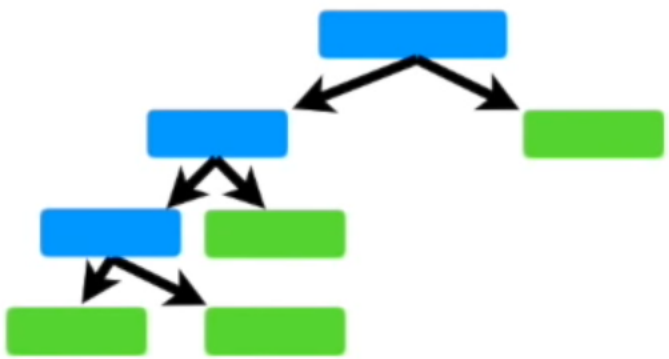
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

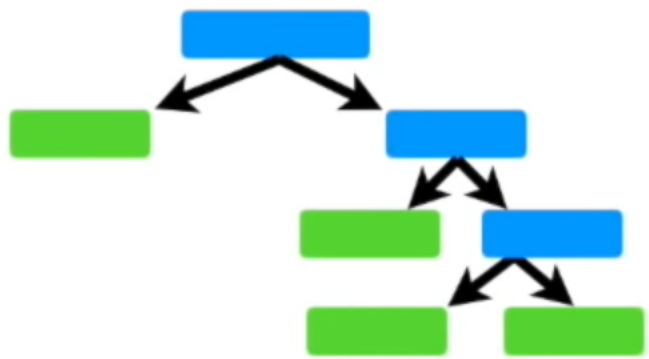
Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.



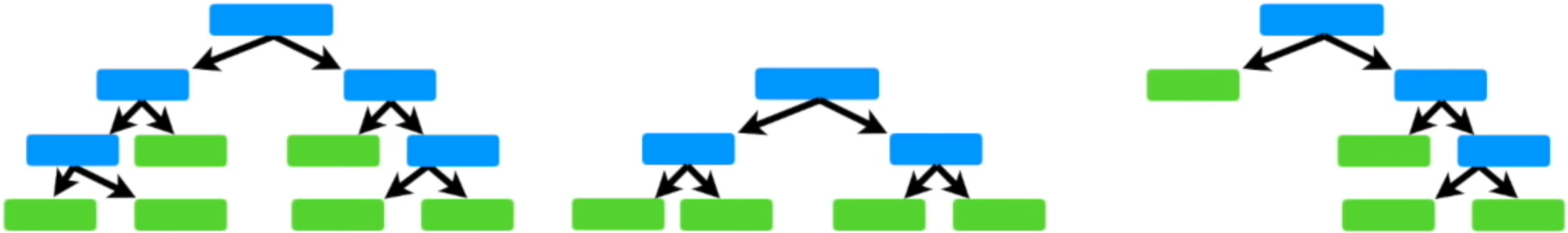




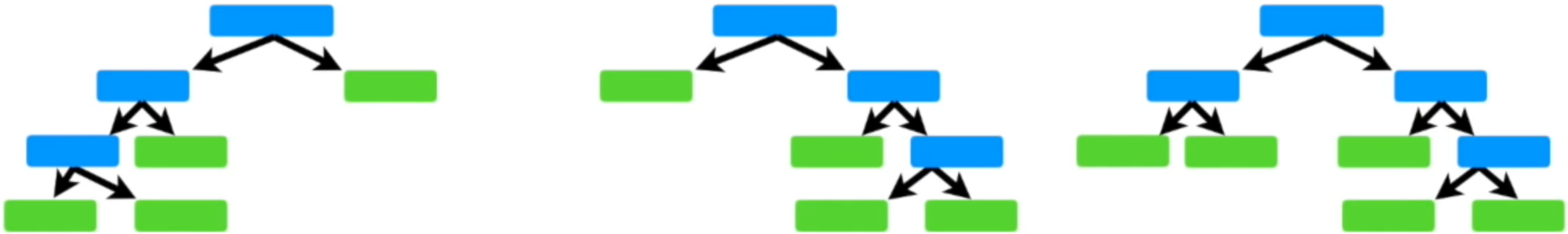




Using a bootstrapped sample and considering only a subset of the variables at each step results in a wide variety of trees.



The variety is what makes random forests more effective than individual decision trees.



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

…and now we want to know if they have heart disease or not.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	

After running the data down all of the trees in the random forest, we see which option received more votes.

Typically, about 1/3 of the original data does not end up in the bootstrapped dataset.

Original Dataset

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

Bootstrapped Dataset

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

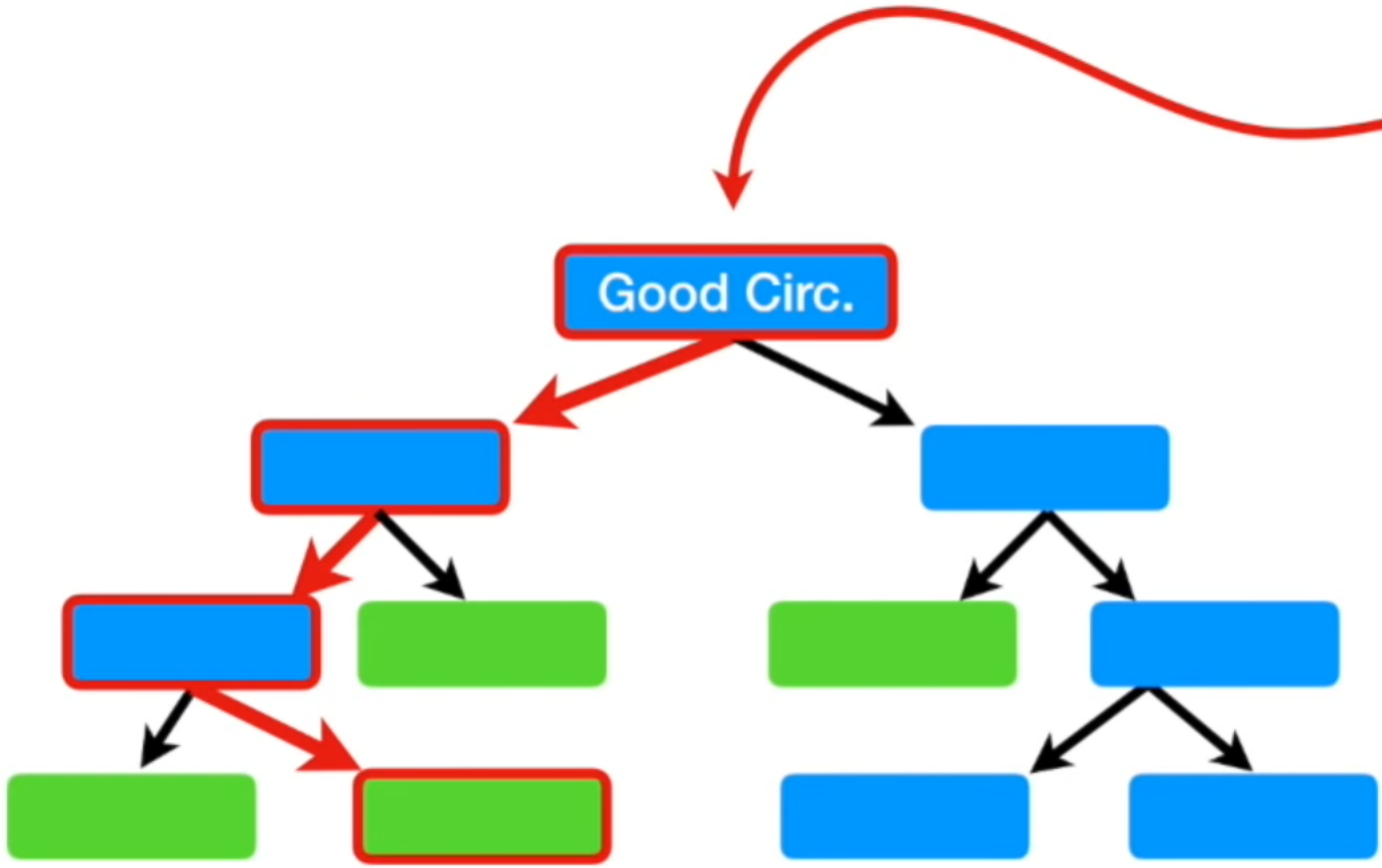
Original Dataset

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

This is called the
"Out-Of-Bag Dataset"

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

…we can run it through and see if it correctly classifies the sample as "No Heart Disease"



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

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Since the label with the most votes wins, it is the label that we assign this Out-of-Bag sample.

In this case, the Out-of-Bag sample is correctly labeled by the Random Forest.

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

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

etc… etc… etc…

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

The proportion of Out-Of-Bag samples that were *incorrectly* classified is the "**Out-Of-Bag Error**"

In other words…

…change the number of variables used per step…

1) Build a Random Forest

2) Estimate the accuracy of a Random Forest.

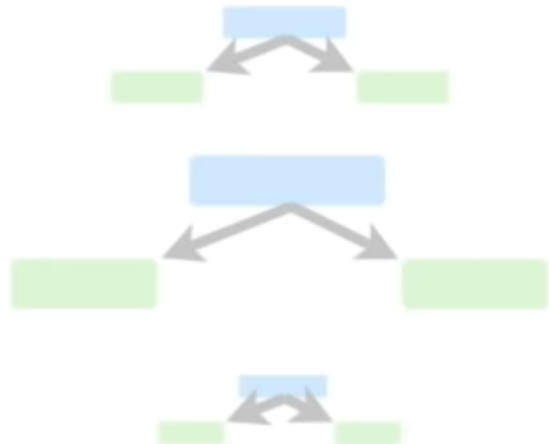
Do this for a bunch of
times and then choose the
one that is most accurate.

AdaBoost

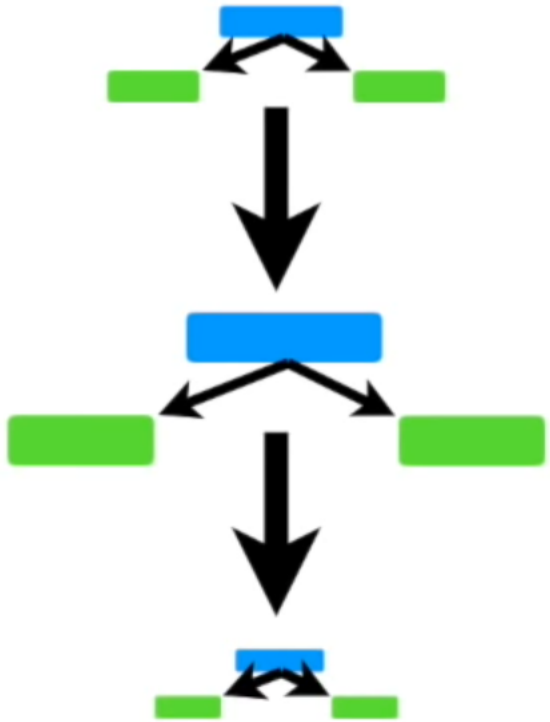
To review, the three ideas behind **AdaBoost** are…

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

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

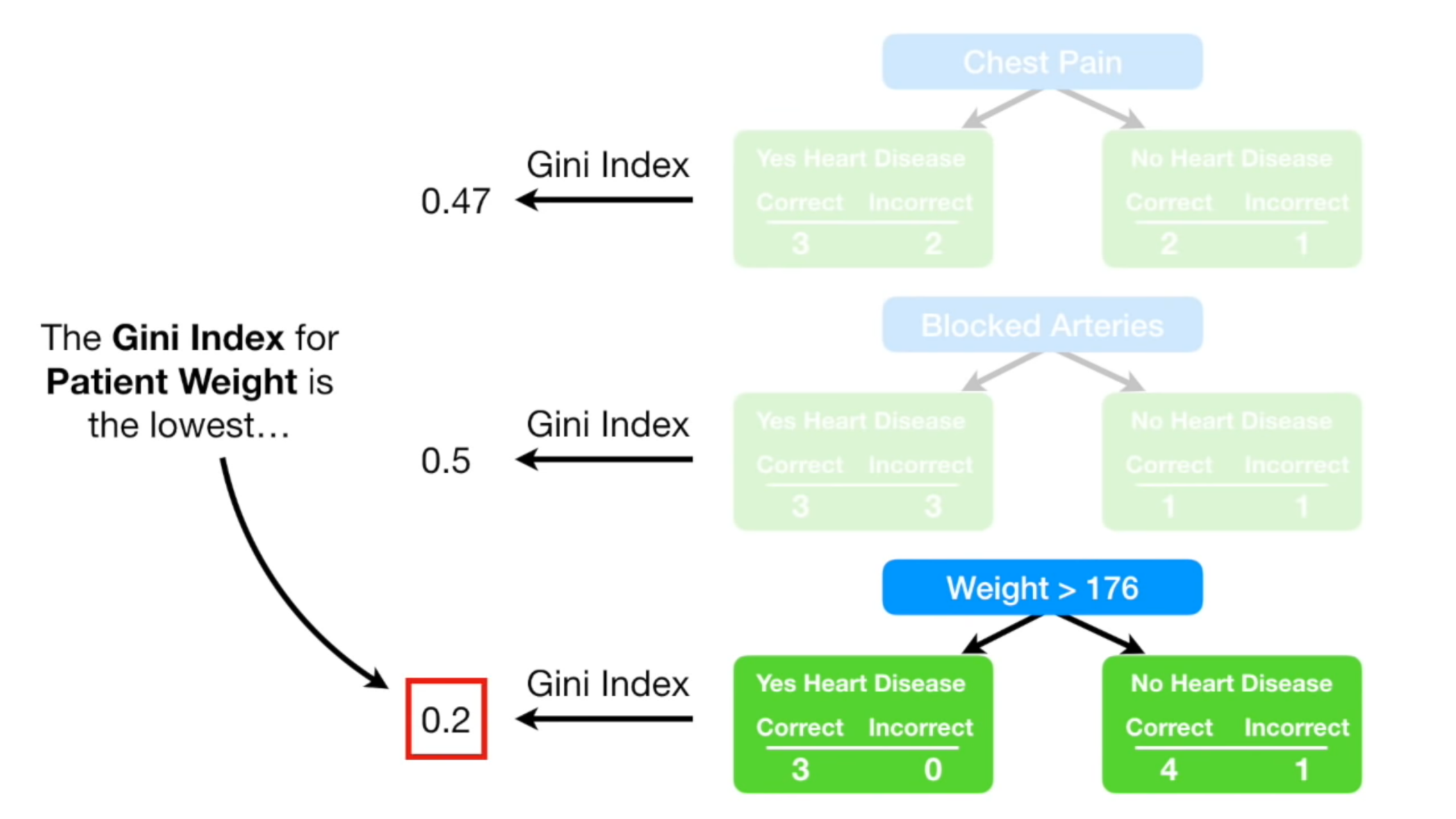


3) Each **stump** is made by taking the previous **stump's** mistakes into account.



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

First, we'll start with some data.


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

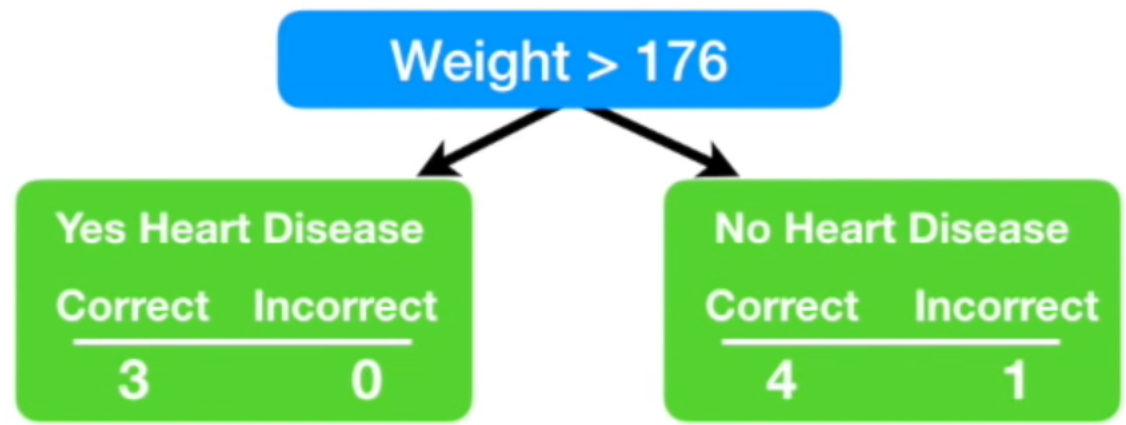
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

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

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

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

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

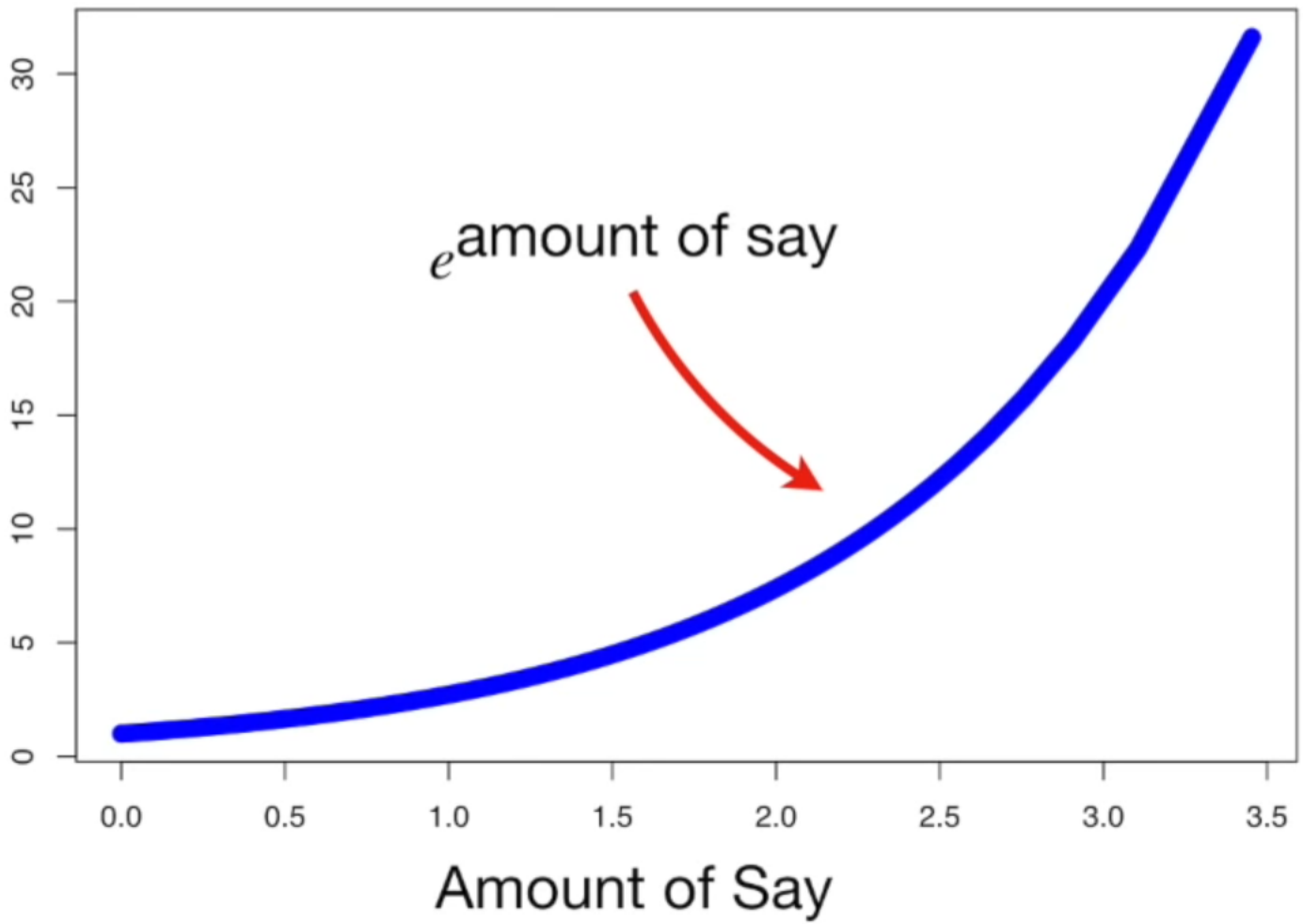


Now we need to learn how to modify the weights so that the next stump will take the errors that the current stump made into account.

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

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

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



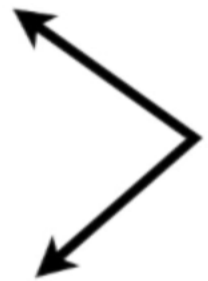
$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

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

$$= \frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64 = 0.33$$

That means the new **Sample Weight** is **0.33**, which is *more* than the old one (**1/8** = **0.125**).

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

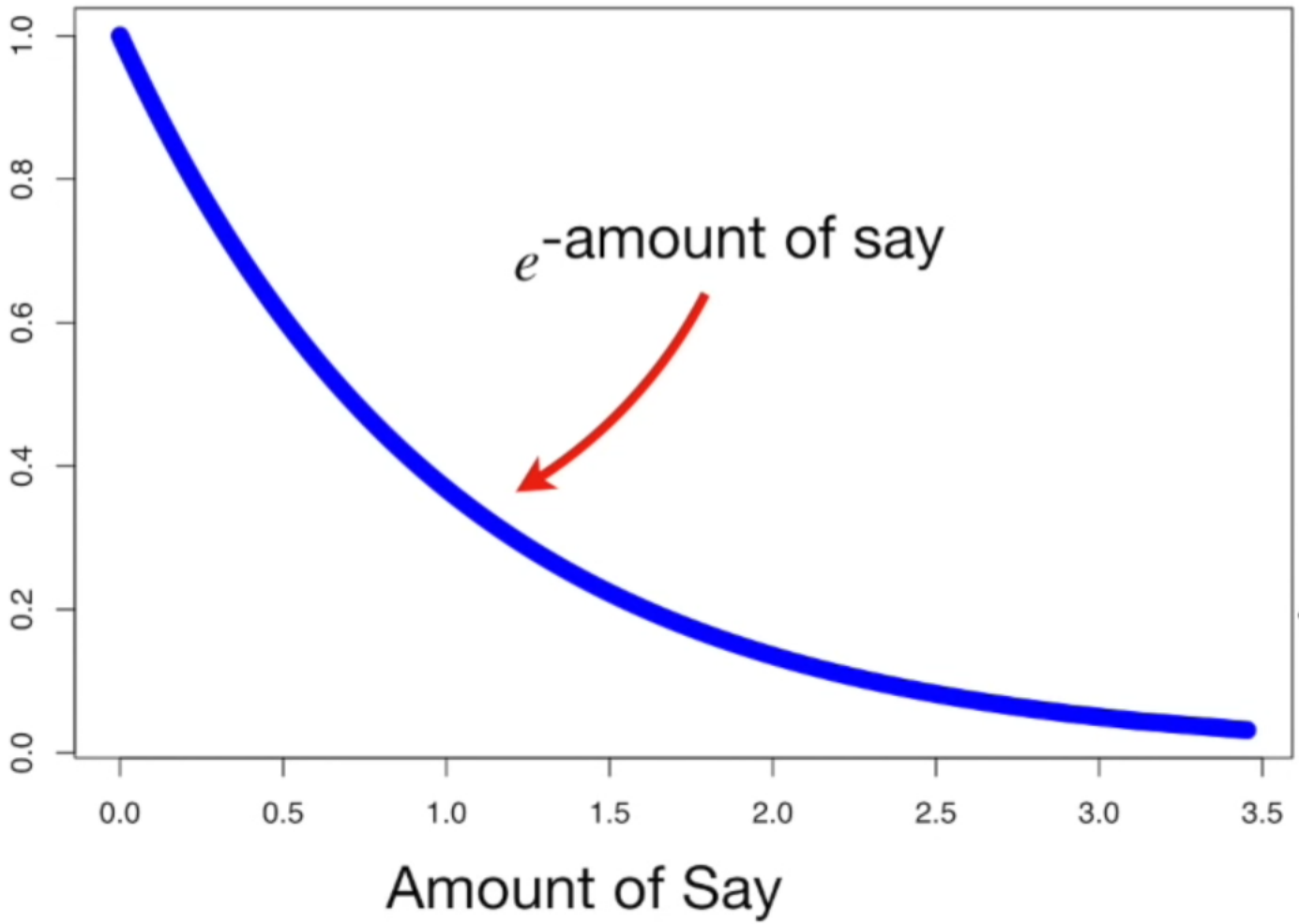


Now we need to *decrease* the **Sample Weights** for all of the *correctly* classified samples.

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

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

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



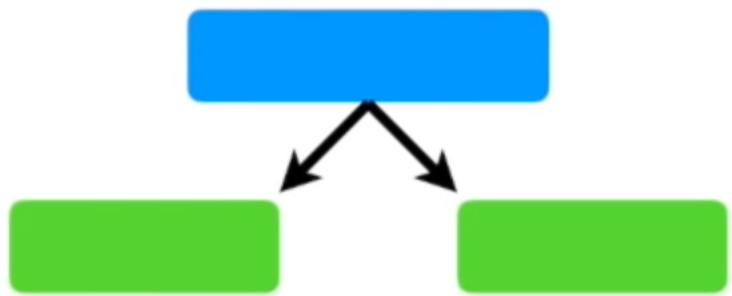
The new **Sample Weight** is **0.05**, which is *less* than the old one (**1/8** = **0.125**).

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

Now, when we add up the **New Sample Weights**, we get **1** (plus or minus a little rounding error).

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

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



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

…and use the new collection of samples.

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

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

Has Heart Disease

Total = 2.7

Total = 1.23

Does Not Have Heart Disease

Amount of Say

0.97

0.32

0.78

0.63

Amount of Say

0.41

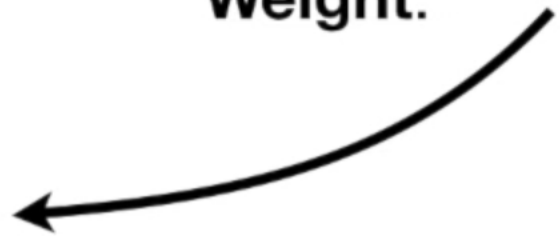
0.82

Gradient Boost

AdaBoost and Gradient Boost

- AdaBoost starts by building a very short tree called a Stump. Gradient Boost starts by making a single leaf instead of tree or Stump. The leaf represent an initial guess (average) of all the samples
- AdaBoost and Gradient Boost trees are based on error made by the previous tree but tree build by Gradient Boost is larger than a Stump
- Like AdaBoost, Gradient Boost scales the trees. But Gradient Boost scales all the trees by the same amount

…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

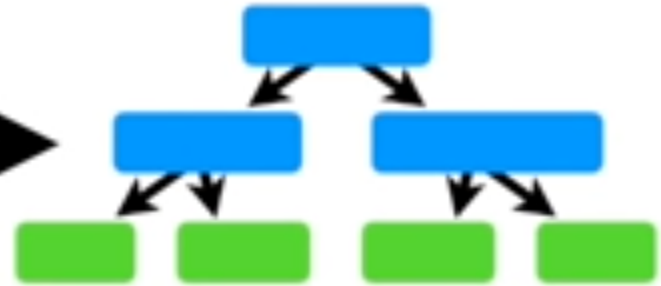
71.2

The first thing we do is calculate the average **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

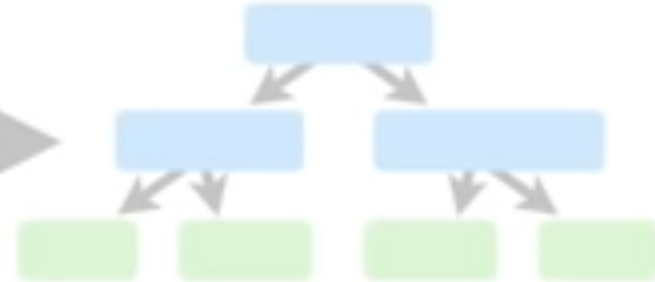


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

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

Average Weight

71.2

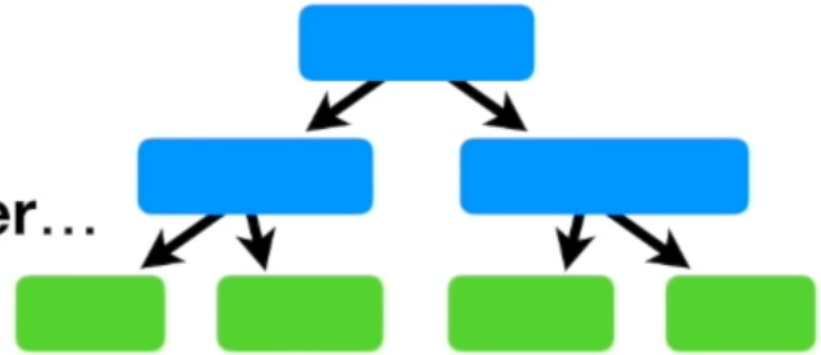


The errors that the previous tree made are the differences between the **Observed Weights** and the **Predicted Weight**, **71.2**.

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

(**Observed Weight** - **Predicted Weight**)

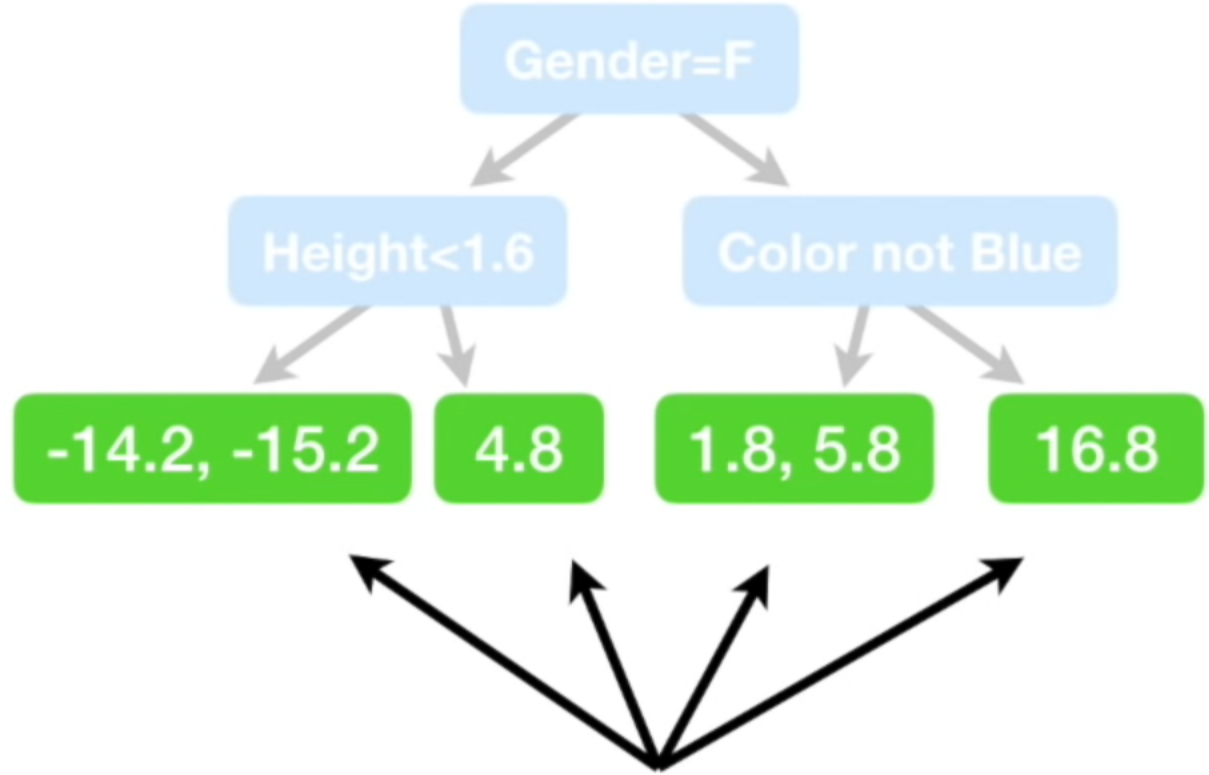
Now we will build a **Tree**, using **Height**, **Favorite Color** and **Gender**…



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**.

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



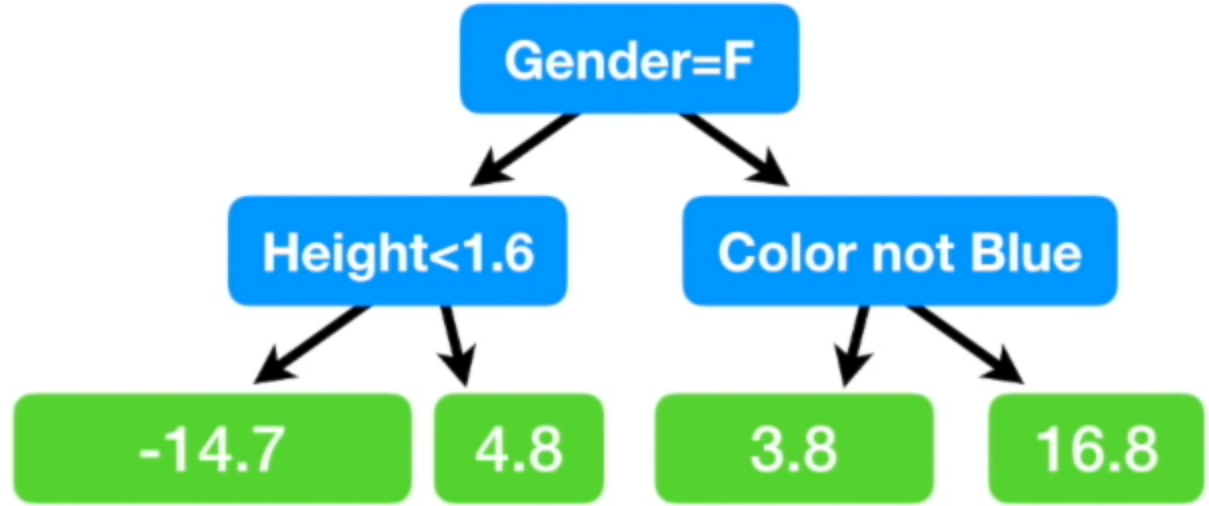
Remember, in this example we are only allowing up to four leaves…

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

Average Weight

71.2

+



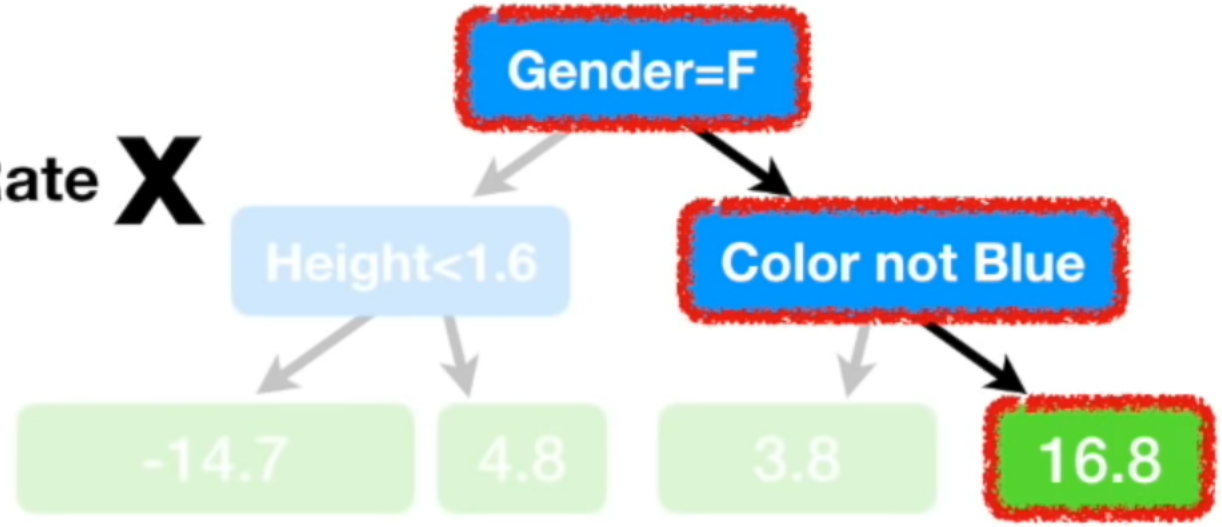
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

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

Average Weight

71.2

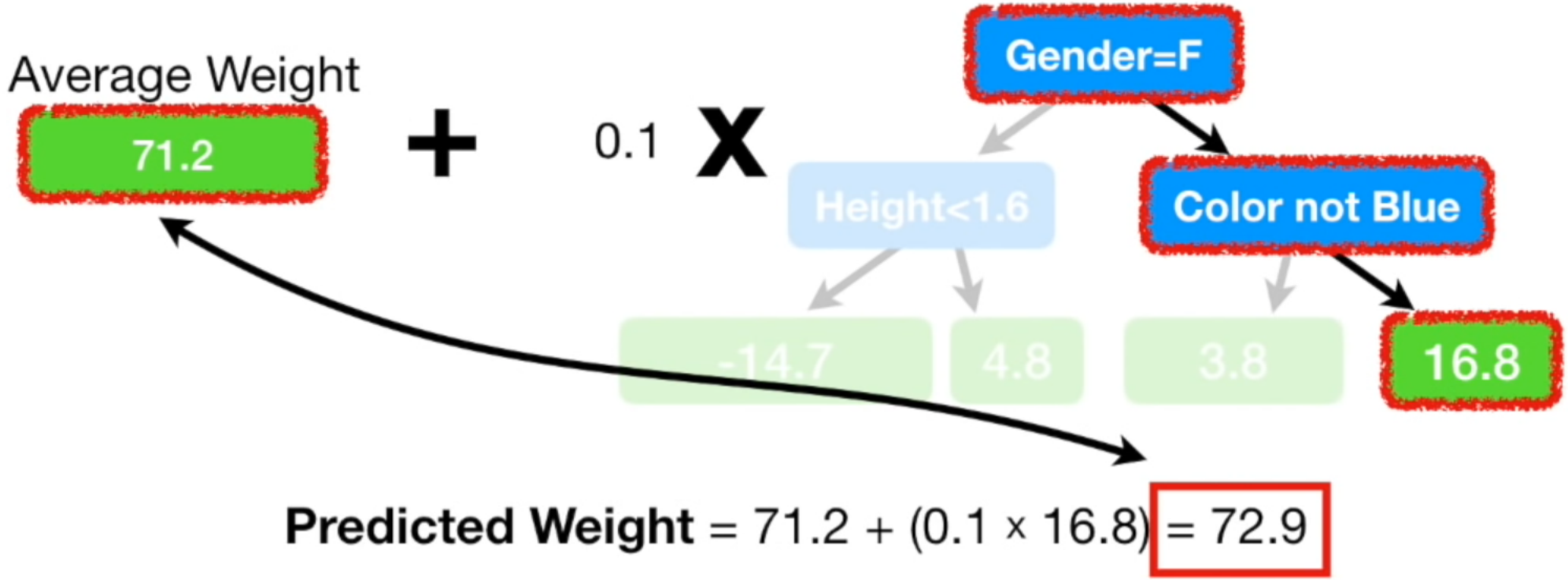
+ Learning Rate X



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

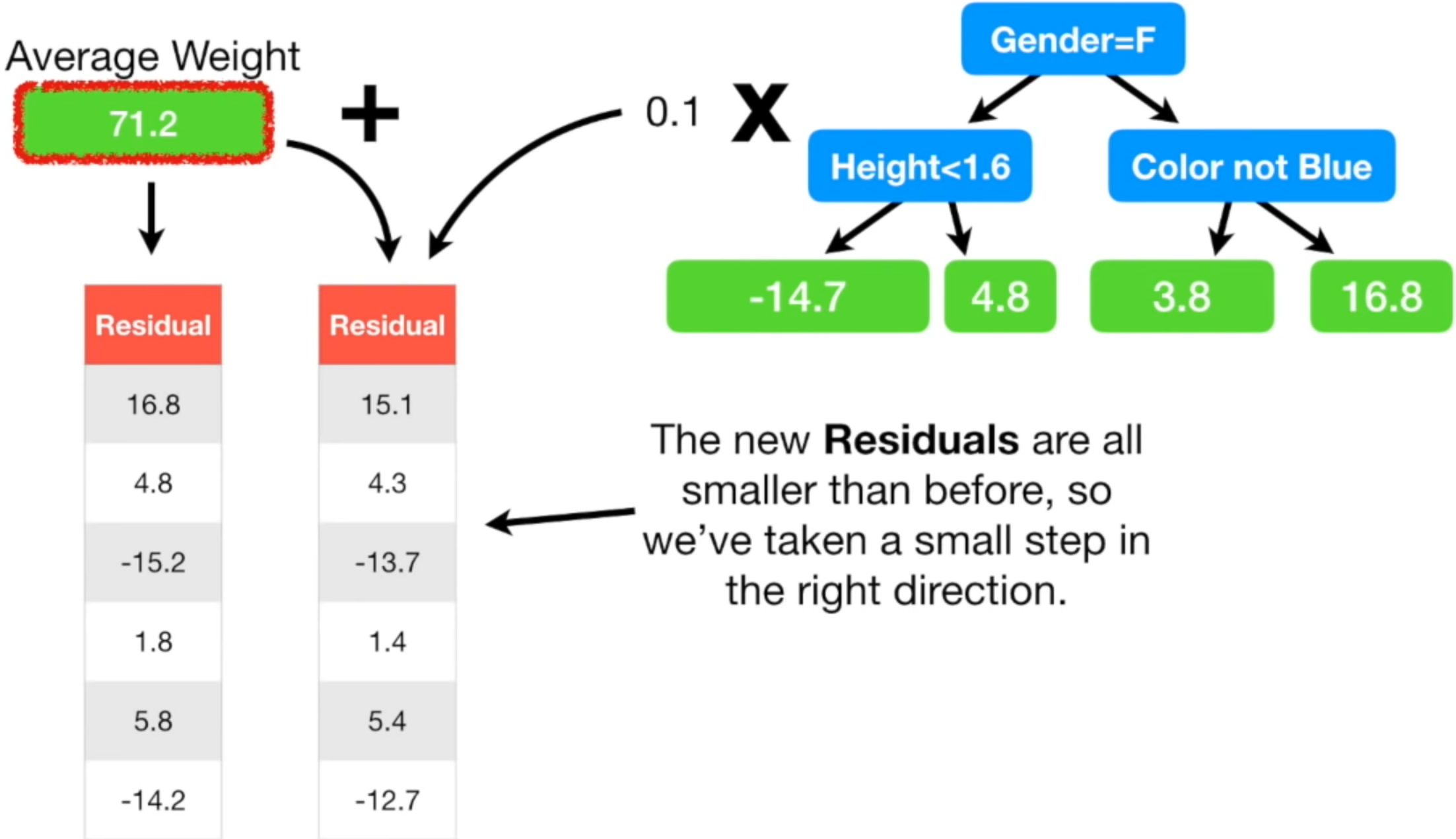
Gradient Boost deals with this problem by using a **Learning Rate** to scale the contribution from the new tree.

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



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

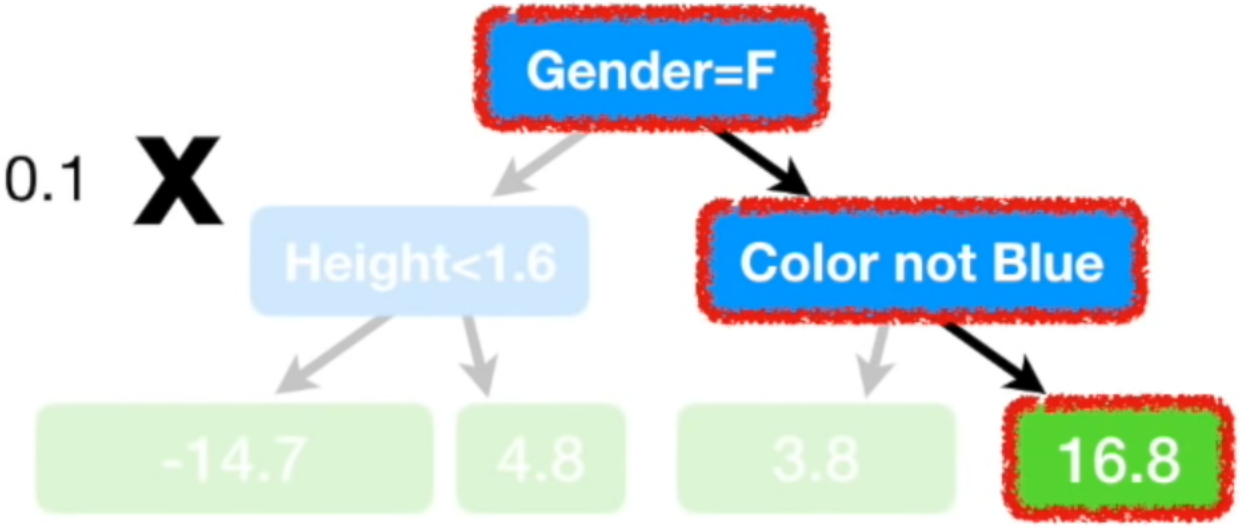
…but it's a little bit better than the **Prediction** made with just the original leaf, which predicted that all samples would weigh **71.2**.


Average Weight
71.2
+
0.1
X
Gender=F
Height<1.6
Color not Blue
-14.7
4.8
3.8
16.8
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7
The new **Residuals** are all smaller than before, so we've taken a small step in the right direction.

Average Weight

71.2

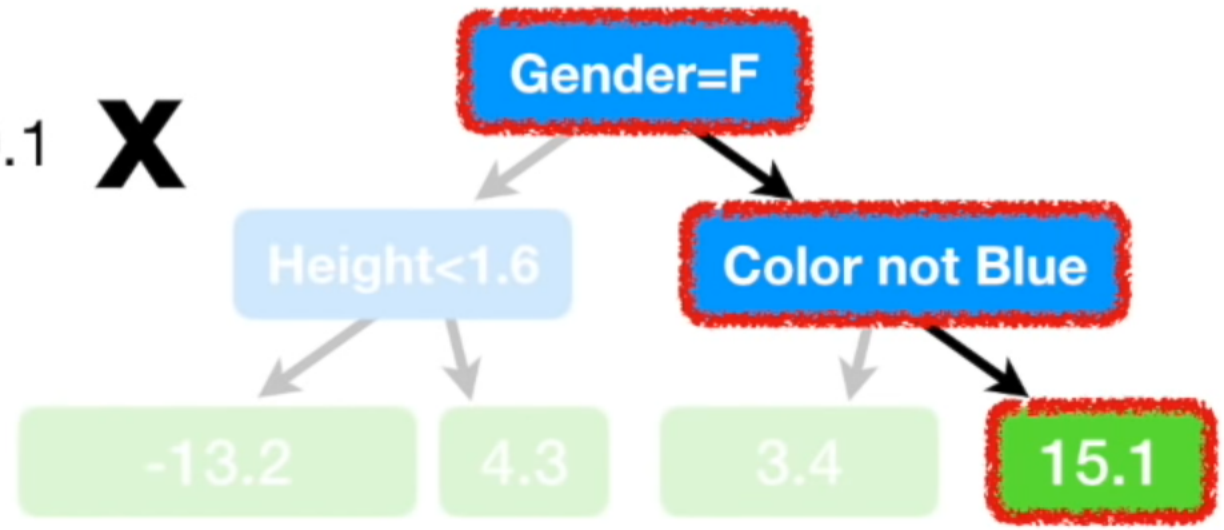
+ 0.1 X



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

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

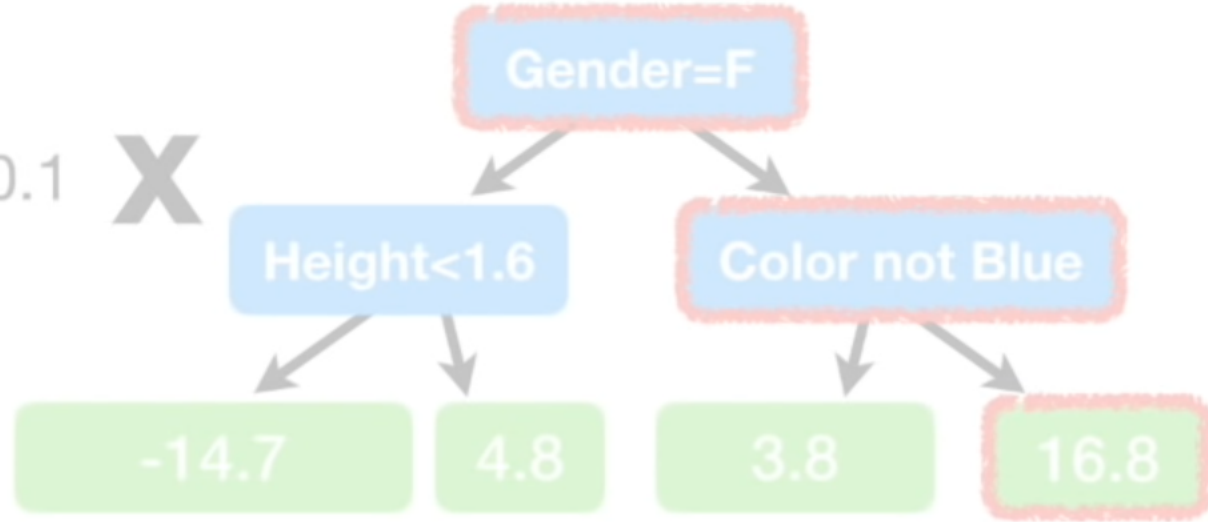
+ 0.1 X



Average Weight

71.2

+ 0.1 X



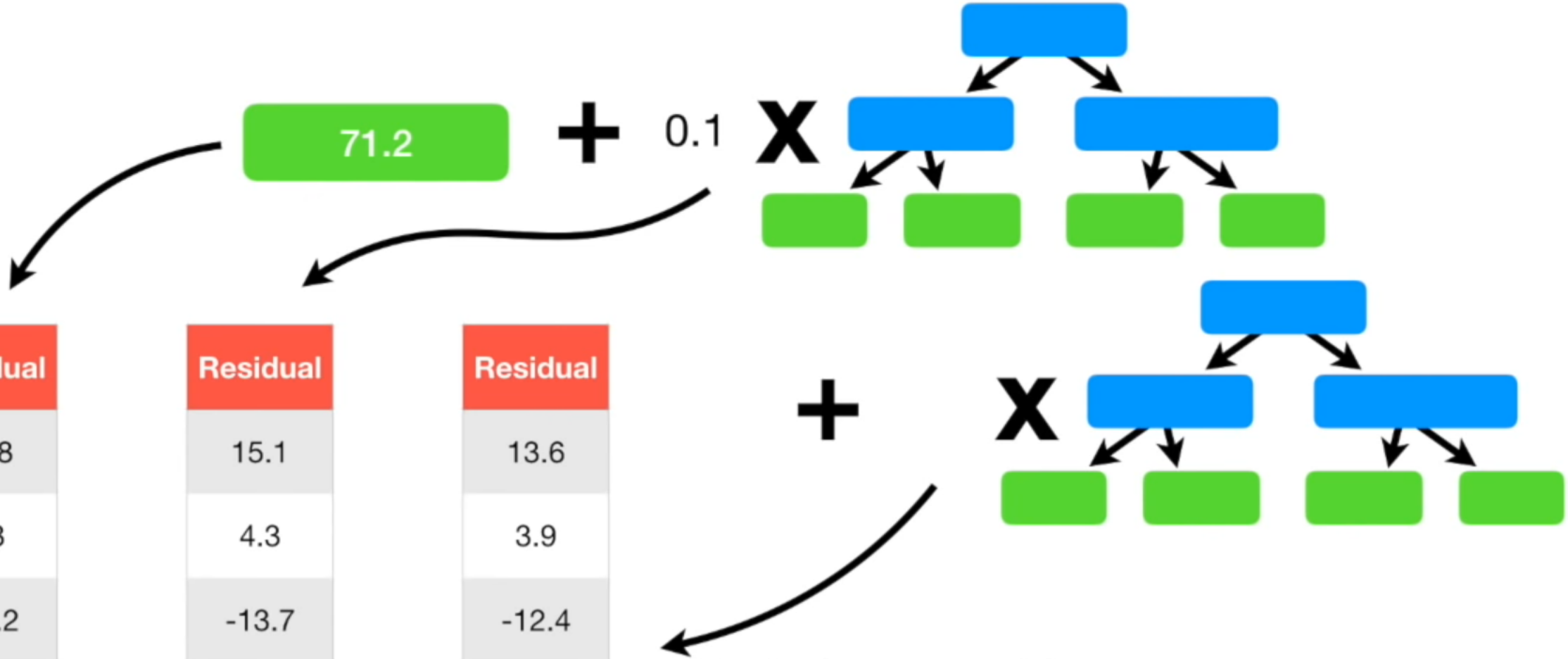
Which is another small step closer to the **Observed Weight**.

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

$$= 74.4$$

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

+ 0.1 X



Residual
16.8
4.8
-15.2
1.8
5.8
-14.2

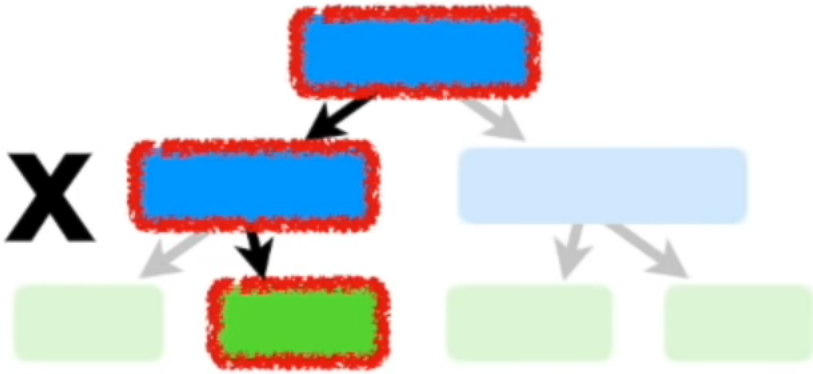
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7

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.

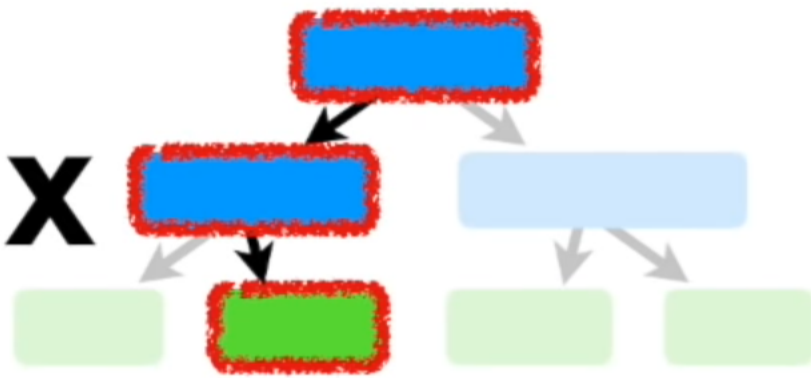
71.2

+ 0.1 X



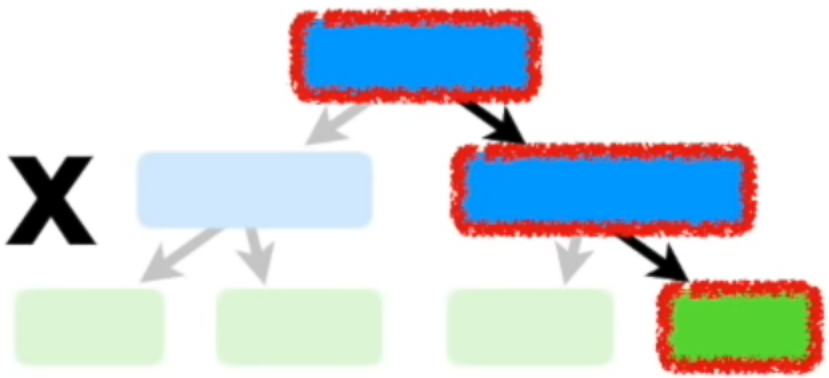
…etc…etc…etc…

+ 0.1 X



Height (m)	Favorite Color	Gender	Weight (kg)
1.7	Green	Female	???

+ 0.1 X



…etc…etc…etc…

Different practice usage in R

- Individual packages / libraries such as randomforest, gbm etc
- Caret package reference
- H2O package reference

References

- LightGBM paper in NeurIPS 2019 - https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

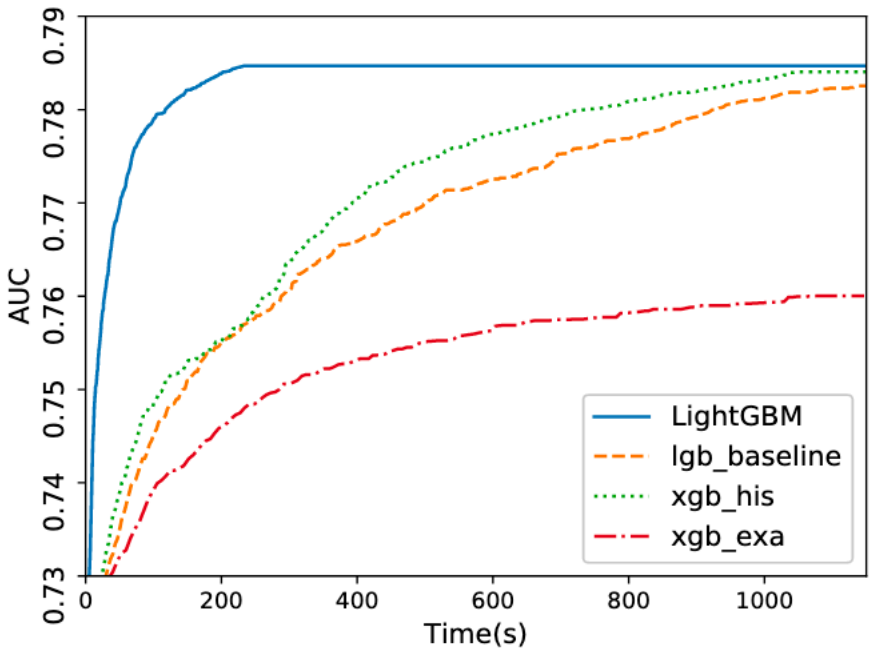


Figure 1: Time-AUC curve on Flight Delay.

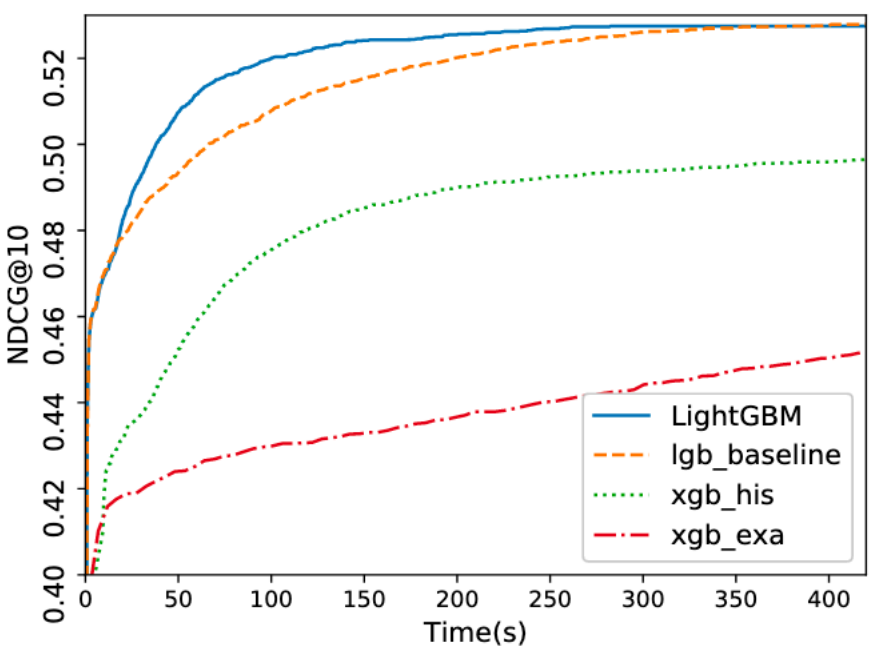


Figure 2: Time-NDCG curve on LETOR.

Q and A