

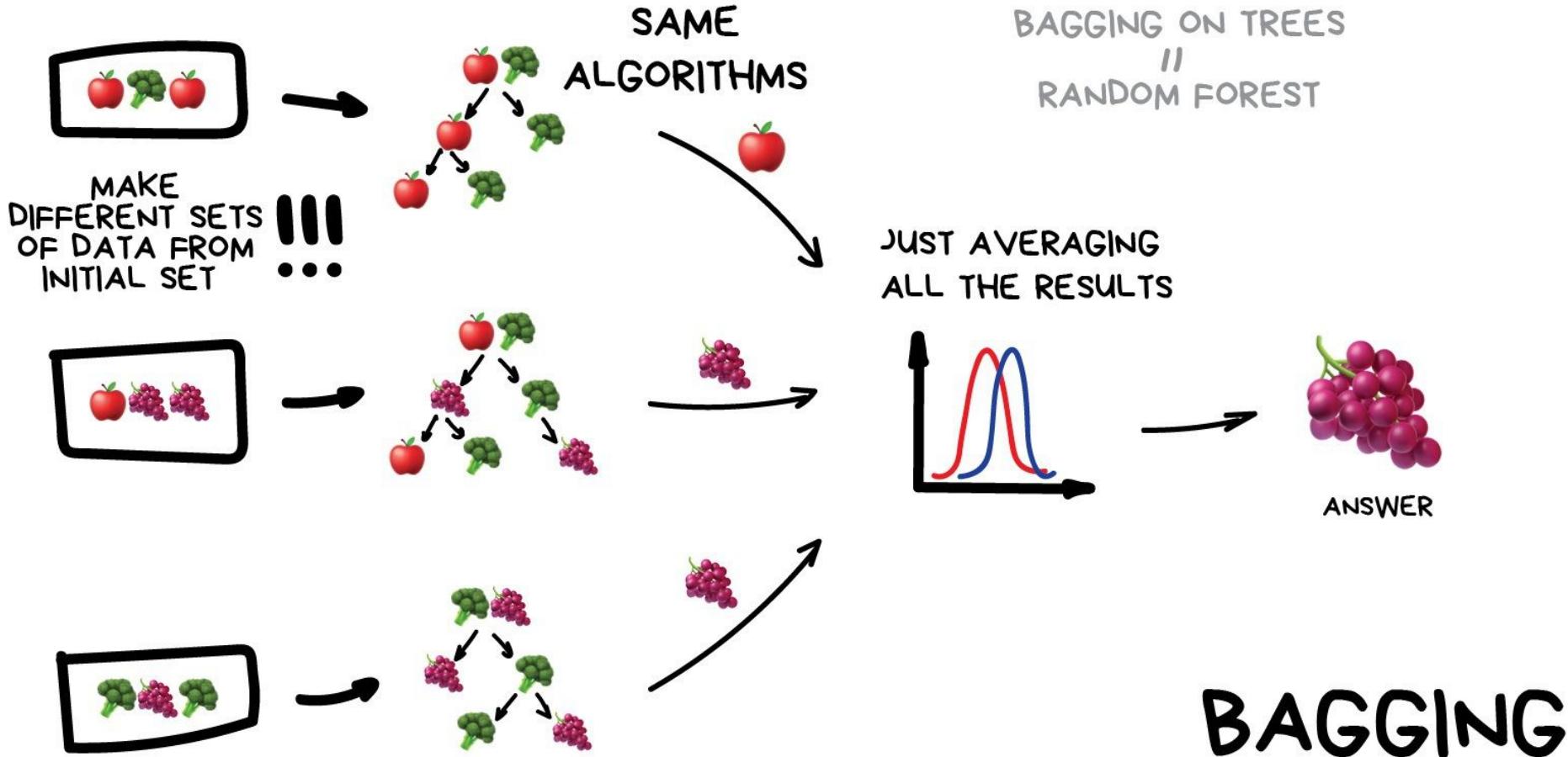
Lesson #07

Ensemble Learning II



ENSEMBLE METHODS

BAGGING & PASTING
BOOSTING
STACKING





How Boosting Works?

Boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

sklearn.ensemble: Ensemble Methods

The `sklearn.ensemble` module includes ensemble-based methods for classification, regression and anomaly detection.

User guide: See the [Ensemble methods](#) section for further details.

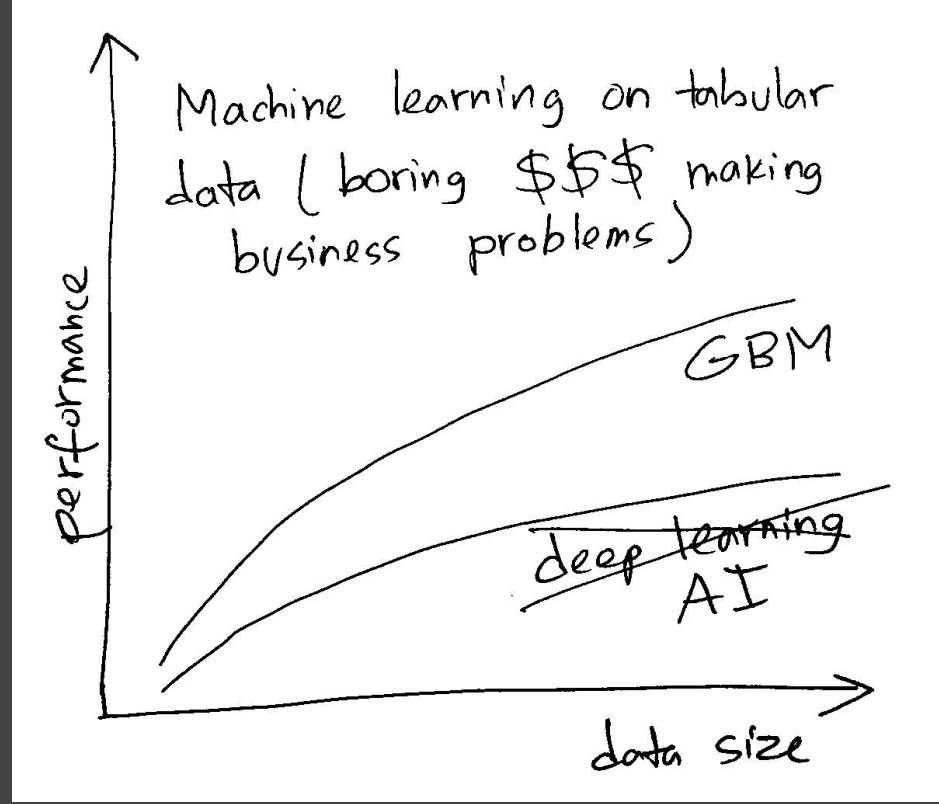
<code>ensemble.AdaBoostClassifier([...])</code>	An AdaBoost classifier.
<code>ensemble.AdaBoostRegressor([base_estimator, ...])</code>	An AdaBoost regressor.
<code>ensemble.BaggingClassifier([base_estimator, ...])</code>	A Bagging classifier.
<code>ensemble.BaggingRegressor([base_estimator, ...])</code>	A Bagging regressor.
<code>ensemble.ExtraTreesClassifier([...])</code>	An extra-trees classifier.
<code>ensemble.ExtraTreesRegressor([n_estimators, ...])</code>	An extra-trees regressor.
<code>ensemble.GradientBoostingClassifier(*[, ...])</code>	Gradient Boosting for classification.
<code>ensemble.GradientBoostingRegressor(*[, ...])</code>	Gradient Boosting for regression.
<code>ensemble.IsolationForest(*[, n_estimators, ...])</code>	Isolation Forest Algorithm.
<code>ensemble.RandomForestClassifier([...])</code>	A random forest classifier.
<code>ensemble.RandomForestRegressor([...])</code>	A random forest regressor.
<code>ensemble.RandomTreesEmbedding([...])</code>	An ensemble of totally random trees.
<code>ensemble.StackingClassifier(estimators[, ...])</code>	Stack of estimators with a final classifier.
<code>ensemble.StackingRegressor(estimators[, ...])</code>	Stack of estimators with a final regressor.
<code>ensemble.VotingClassifier(estimators, *[, ...])</code>	Soft Voting/Majority Rule classifier for unfitted estimators.
<code>ensemble.VotingRegressor(estimators, *[, ...])</code>	Prediction voting regressor for unfitted estimators.
<code>ensemble.HistGradientBoostingRegressor([...])</code>	Histogram-based Gradient Boosting Regression Tree.
<code>ensemble.HistGradientBoostingClassifier([...])</code>	Histogram-based Gradient Boosting Classification Tree.



TABULAR DATA YOU HAVE



GBM USE YOU MUST

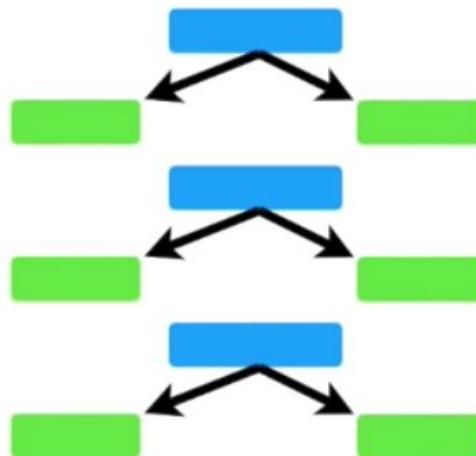


Benchmark:
<https://github.com/szilard/GBM-perf>

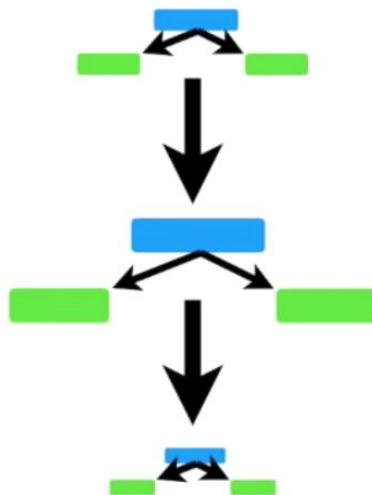
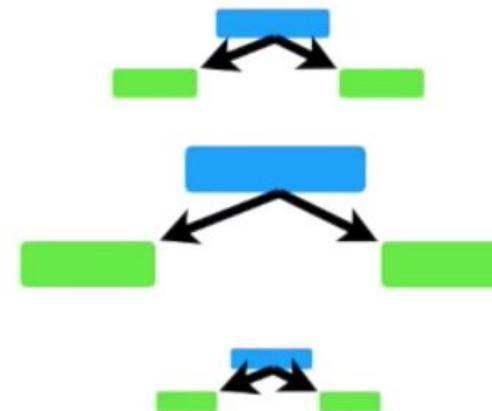
Slides: http://bit.do/slides_gbm
Video: http://bit.do/video_gbm

Adaptive Boosting (AdaBoosting)

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



2. Some **stumps** get more say in the classification than others
3. Each **stump** is made by taking the previous stump's mistake into account



“The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness. AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns.”

“This means that samples that are difficult to classify receive increasing larger weights until the algorithm identifies a model that correctly classifies these samples.”

Jason Brownlee

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

At the start, all samples get the same weight...

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

...and that makes the samples all equally important.

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

Chest Pain

Yes Heart Disease

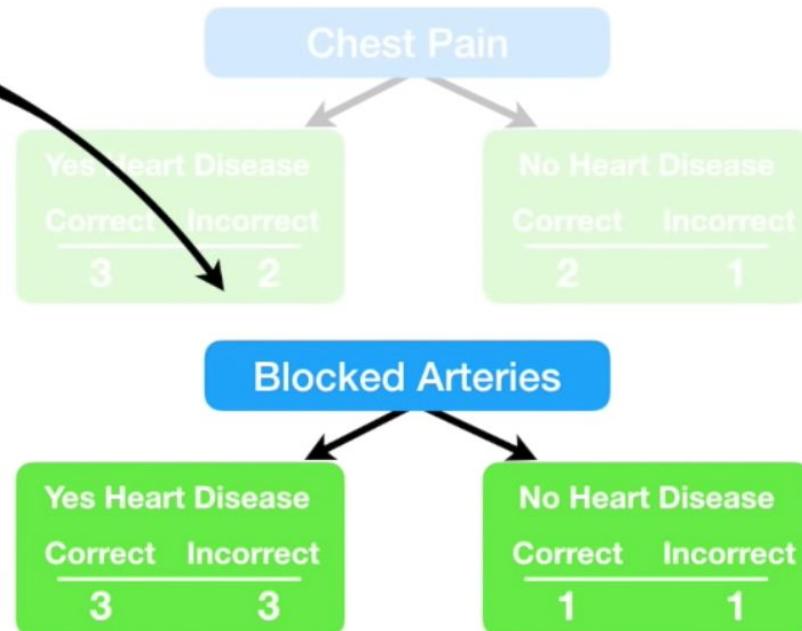
Correct	Incorrect
3	2

No Heart Disease

Correct	Incorrect
2	1

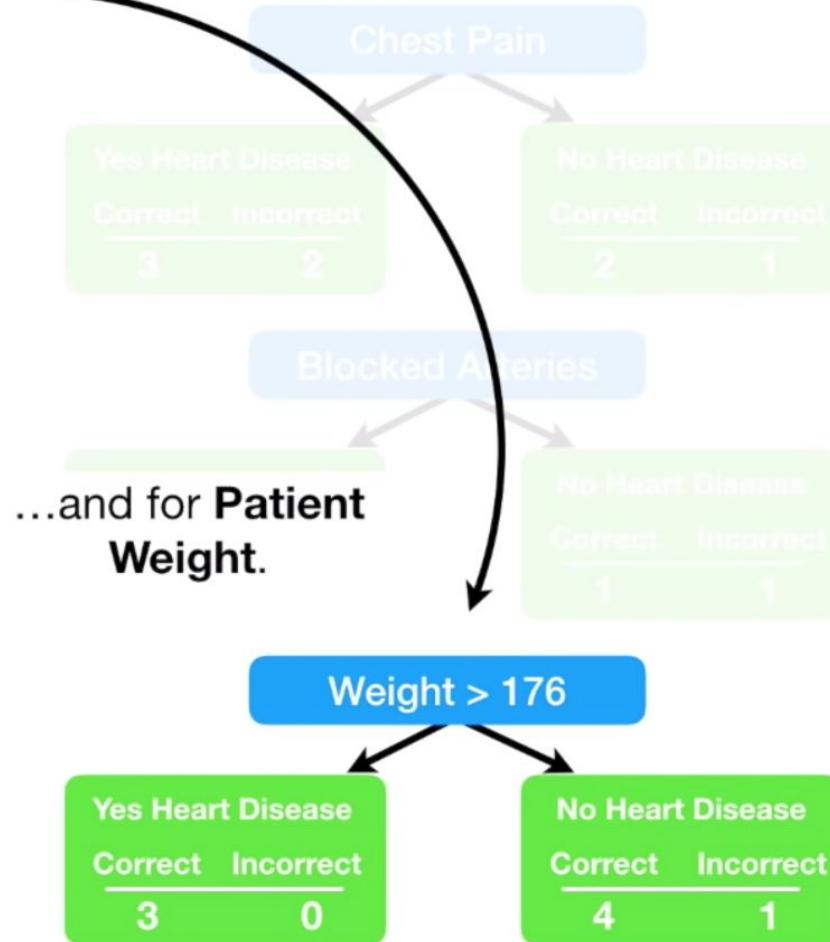
We start by seeing how well
Chest Pain classifies the
samples.

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



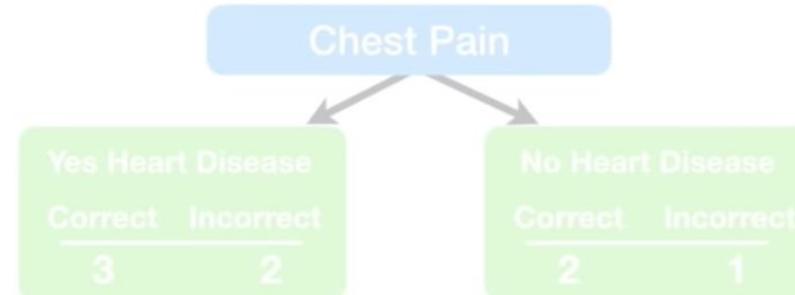
Now we do the same thing
for **Blocked Arteries...**

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



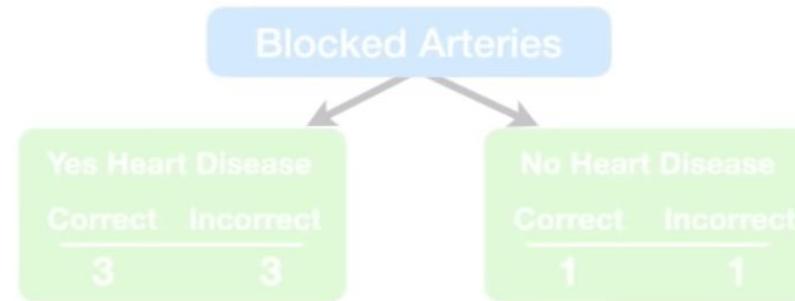
$$Gini(x) = 1 - \sum_{i=1}^c P(x_i)^2$$

Gini Index
0.47

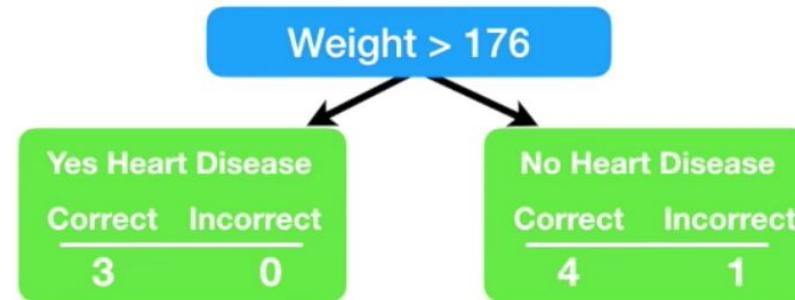


The **Gini Index** for
Patient Weight is
the lowest...

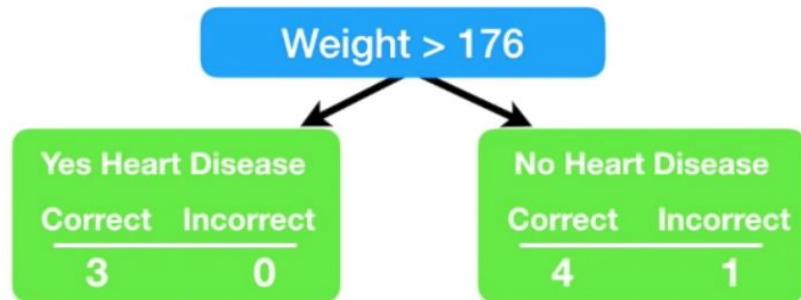
Gini Index
0.5



Gini Index
0.25

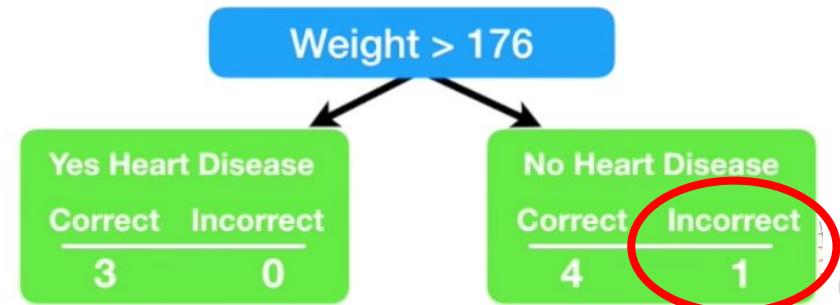


Now we need to determine how much say this stump will have in the final classification.



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

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



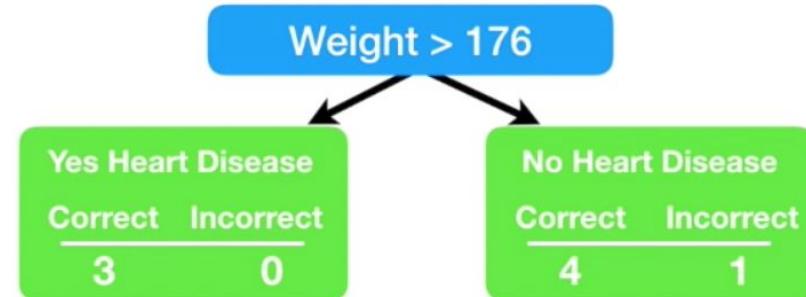
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

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



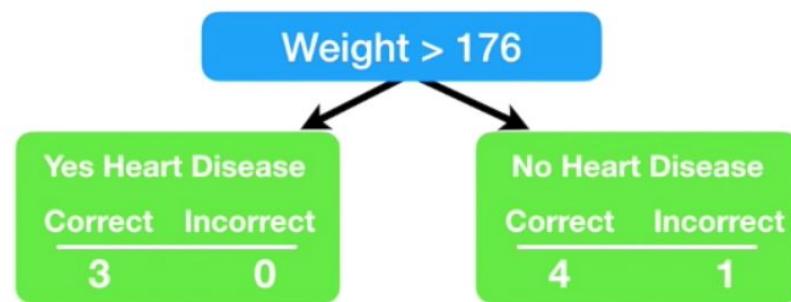
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

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



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

NOTE: Because all of the **Sample Weights** add up to **1**, **Total Error** will always be between **0**, for a perfect stump, and **1**, for a horrible stump.



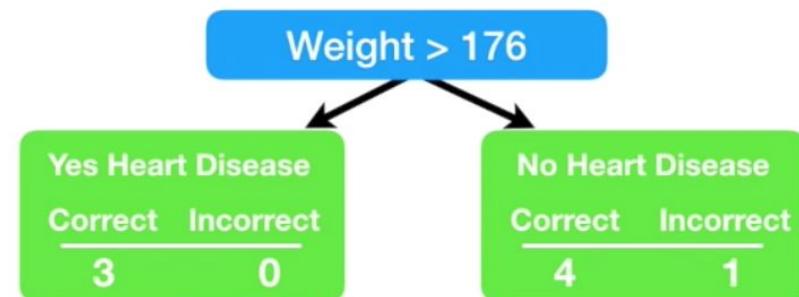
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

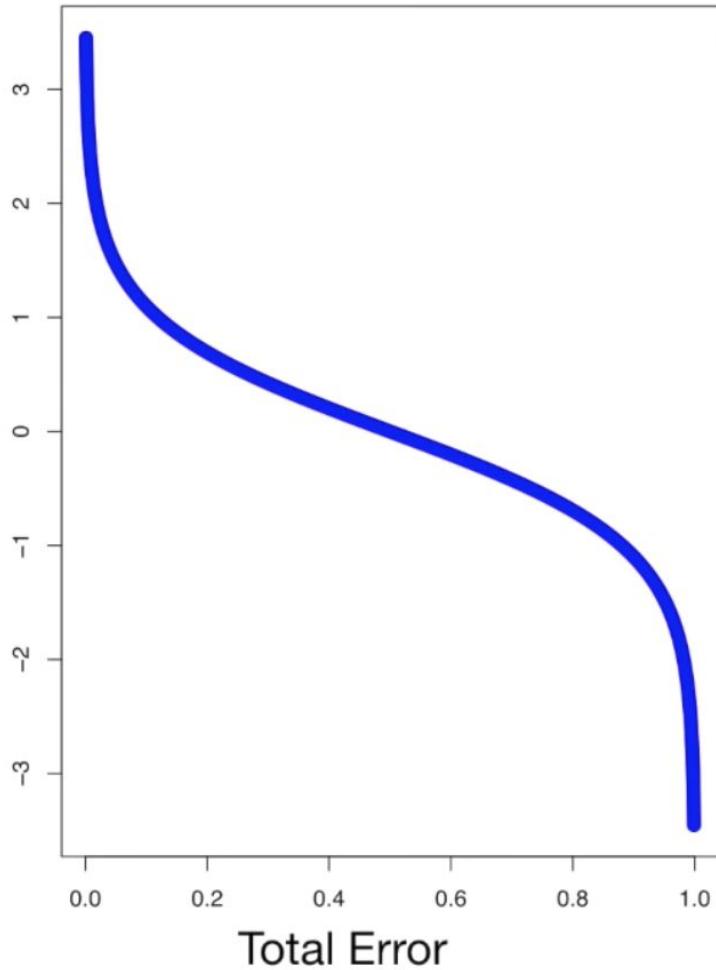
We use the **Total Error** to determine **Amount of Say** this stump has in the final classification with the following formula:



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

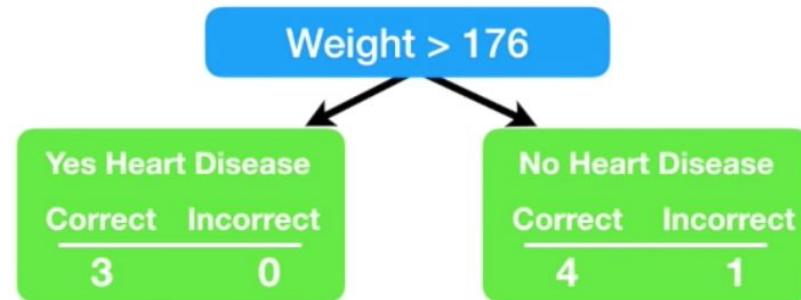
Learning Rate

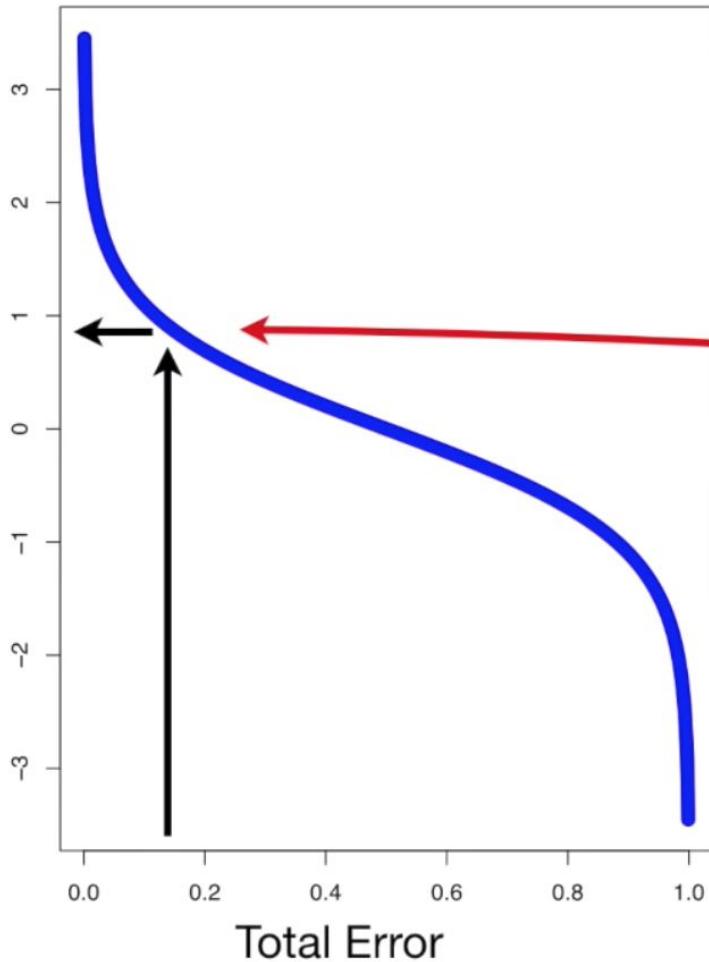




We can draw a graph of the **Amount of Say** by plugging in a bunch of numbers between **0** and **1** for **Total Error**.

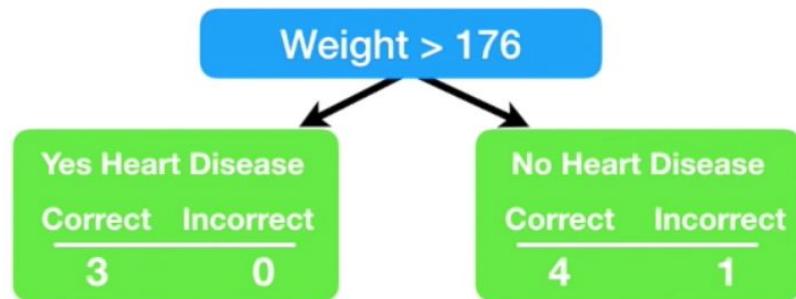
$$\text{Amount of Say} = \frac{1}{2} \log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$





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

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

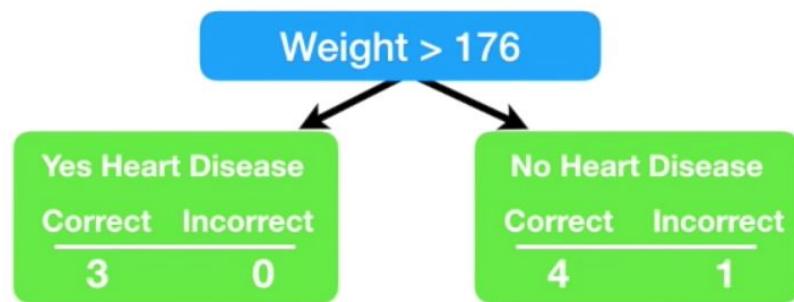


Now we need to learn how to modify the weights so that the next stump will take the errors that the current stump made into account.



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

When we created this stump, all of the **Sample Weights** were the same...



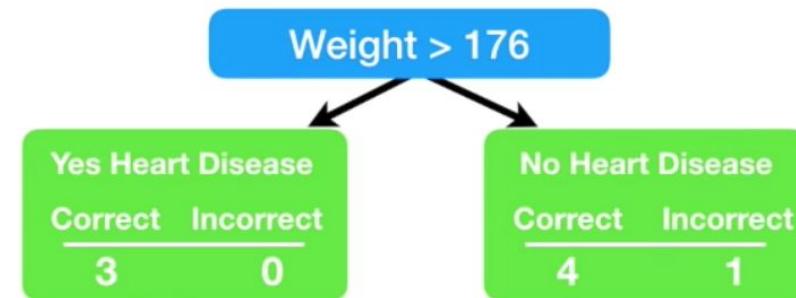
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

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



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

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

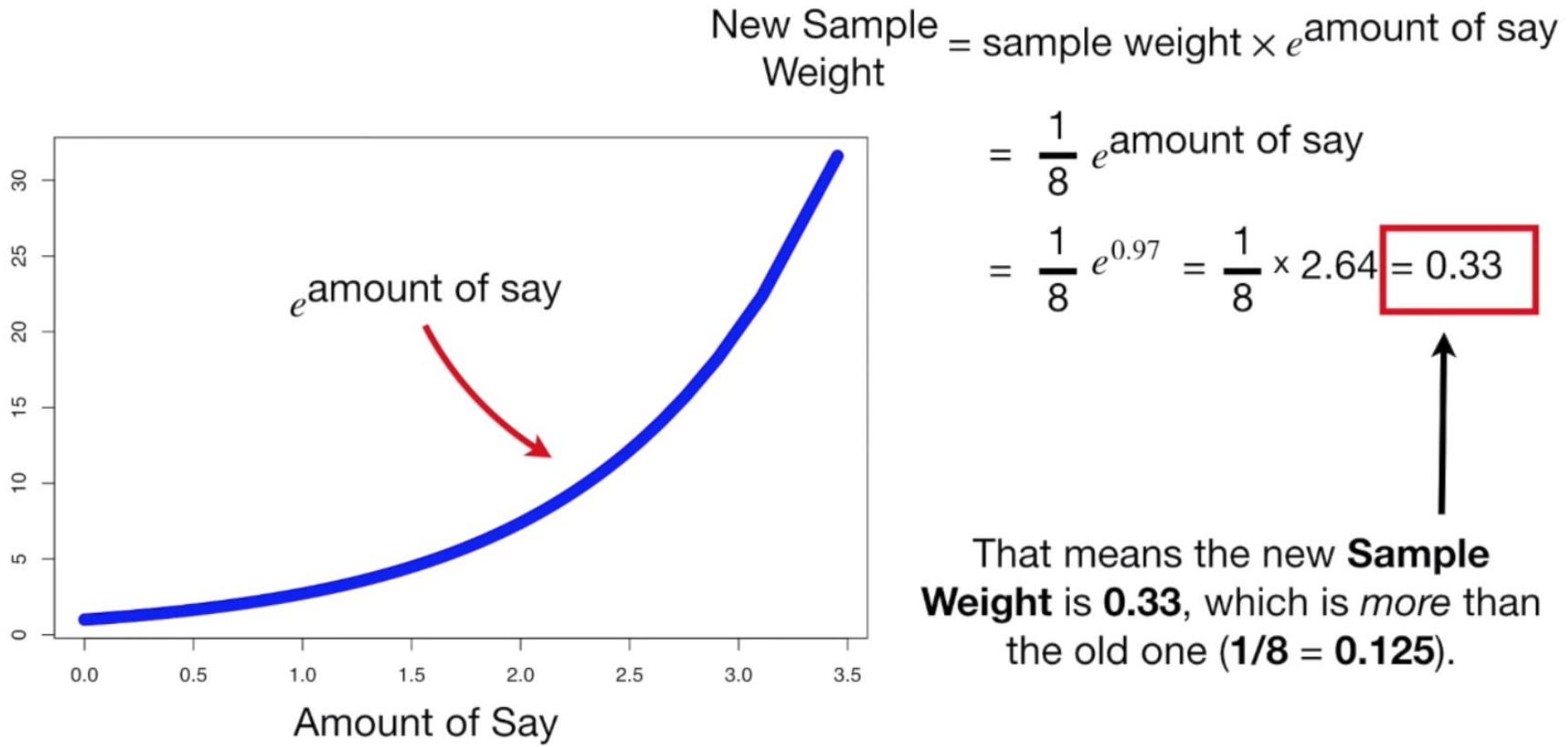



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

New Sample Weight = 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.



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.

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

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

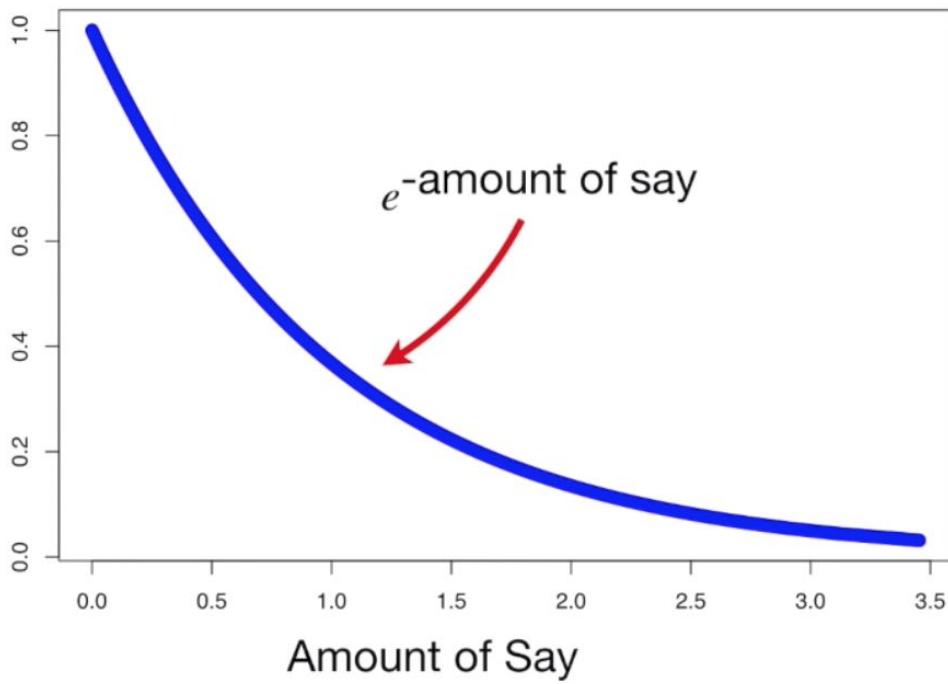


This is the formula we will use to decrease the **Sample Weights.**

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

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

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



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

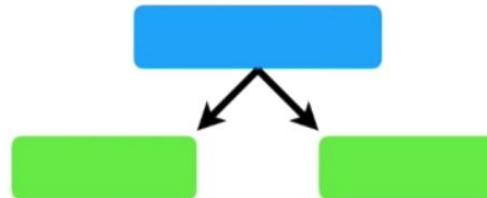
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes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

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



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

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



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

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

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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

...and we see where that number falls when we use the **Sample Weights** like a distribution.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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



If the number is between **0** and **0.07**, then we would put this sample into the new collection of samples...

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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



...and if the number is between **0.07** and **0.14** (**0.07 + 0.07 = 0.14**), then we would put this sample into the new collection of samples...

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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



...and if the number is between **0.14** and **0.21** (**0.14 + 0.07 = 0.21**), then we would put this sample into the new collection of samples...

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

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

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

...and if the number is between **0.21** and **0.70** (**0.21 + 0.49 = 0.70**), then we would put this sample into the new collection of samples...



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

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

For example, imagine
the first number I
picked was **0.72...**

...then I would put this
sample into my new
collection of samples...



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

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No

Then I pick another random number and get **0.42...**

...and I would put this sample into my new collection of samples...



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

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

Then I pick **0.83...**

...and I would put this sample
into my new collection of
samples...



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

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

Then I pick **0.51...**

...and I would put this sample
into my new collection of
samples...



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

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



NOTE: This is the second time that we have added this particular sample to the new collection of samples.

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	We then continue to pick random numbers and add samples to the new collection until we the new collection is the same size as the original.			
Yes				
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

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

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

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

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

Now we get rid of the original 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

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

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



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

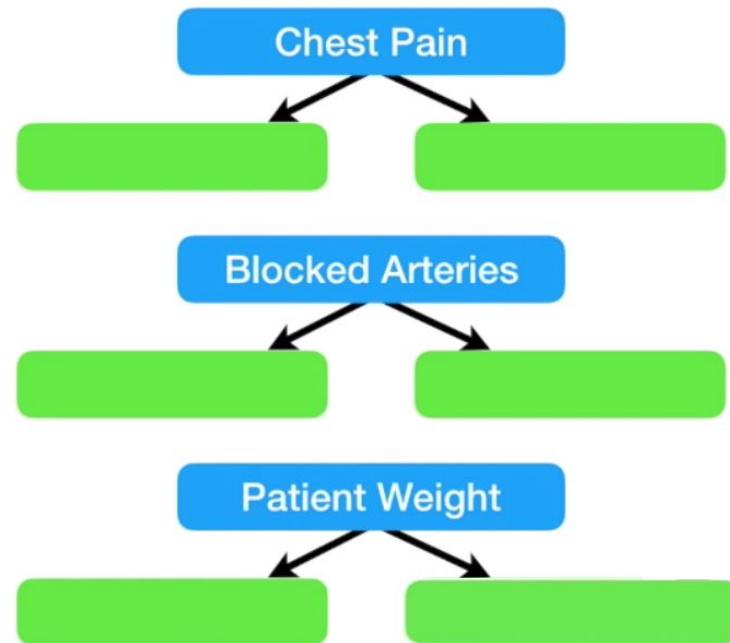
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8



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

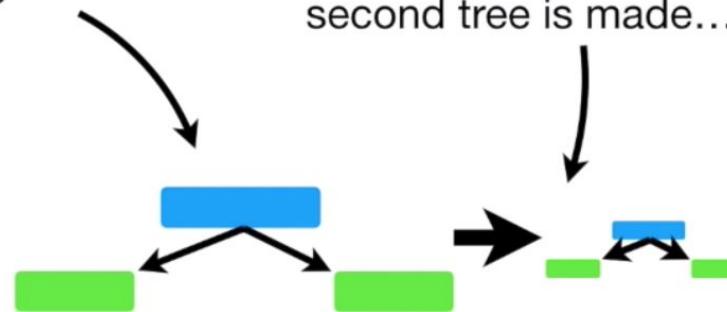
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

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



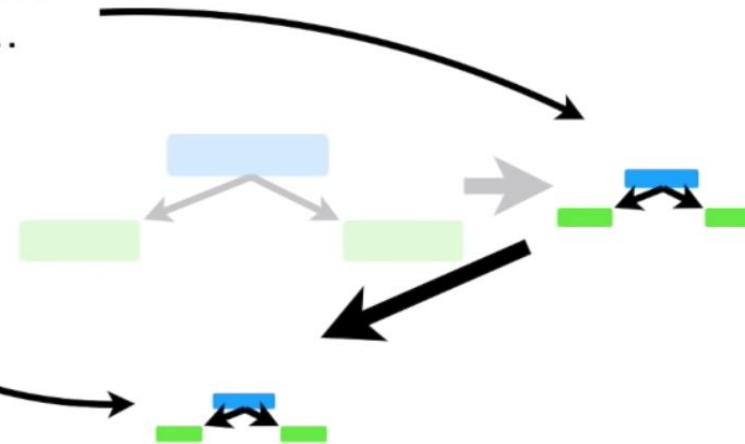
So that is how the errors
that the first tree
makes...

...influence how the
second tree is made...

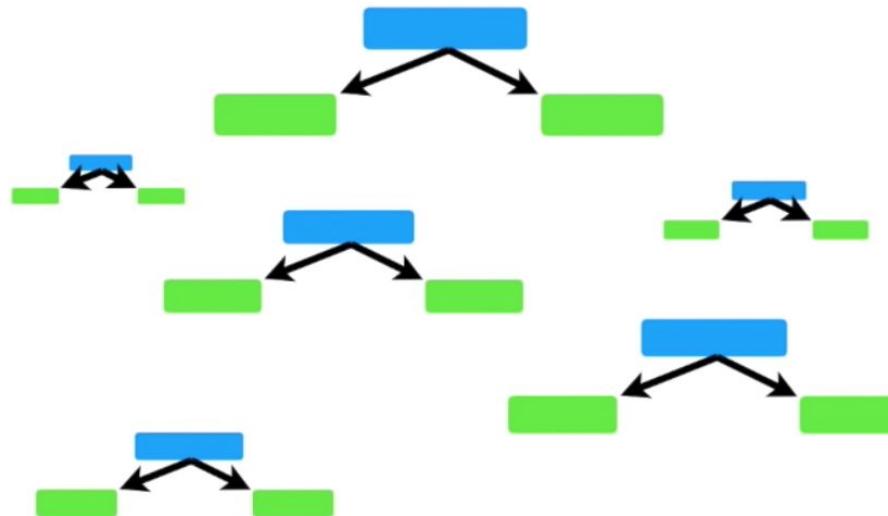


...and the errors that the second tree makes...

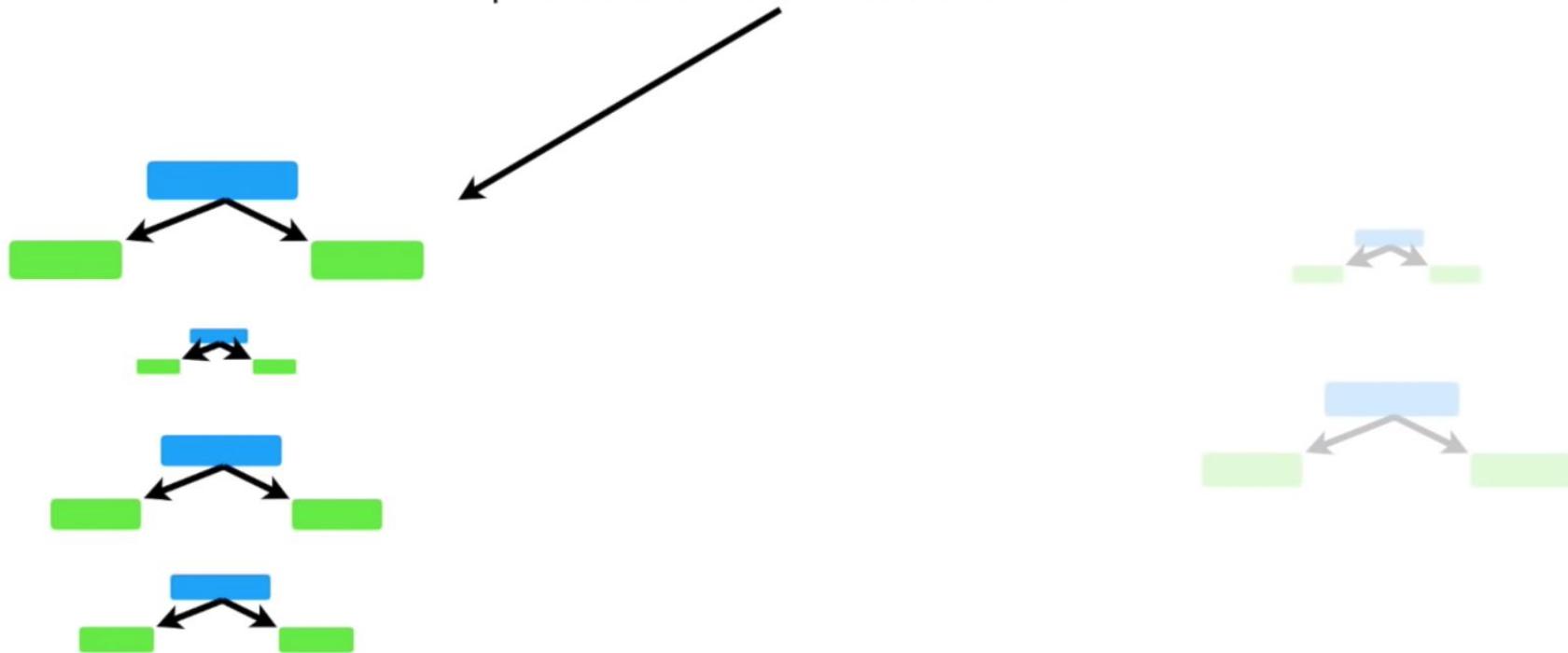
...influence how the
third tree is made.



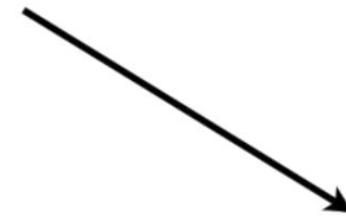
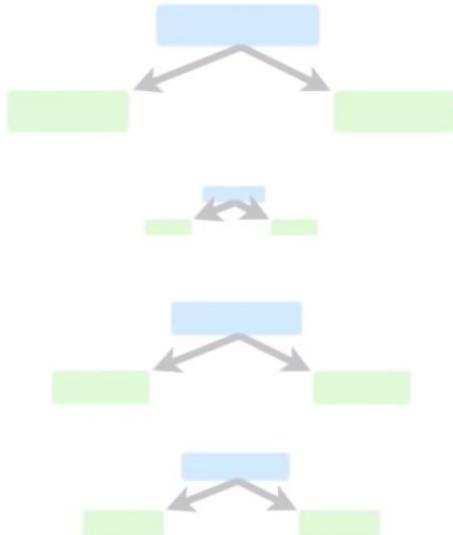
Now we need to talk about how a forest of stumps created by **AdaBoost** makes classifications...



Imagine that these stumps classified a patient as **Has Heart Disease...**



...and these stumps classified the patient as **Does Not Have Heart Disease.**



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

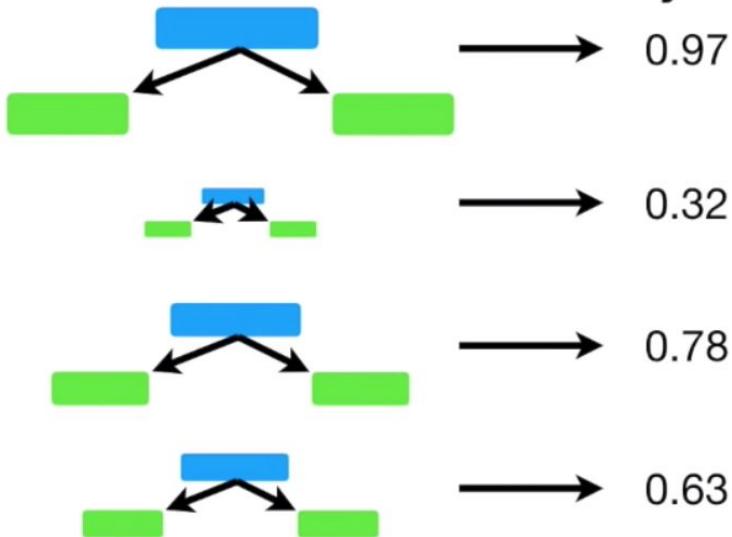
Has Heart Disease

Total = 2.7

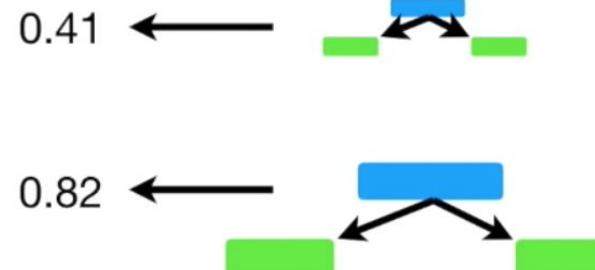
Total = 1.23

Does Not Have Heart Disease

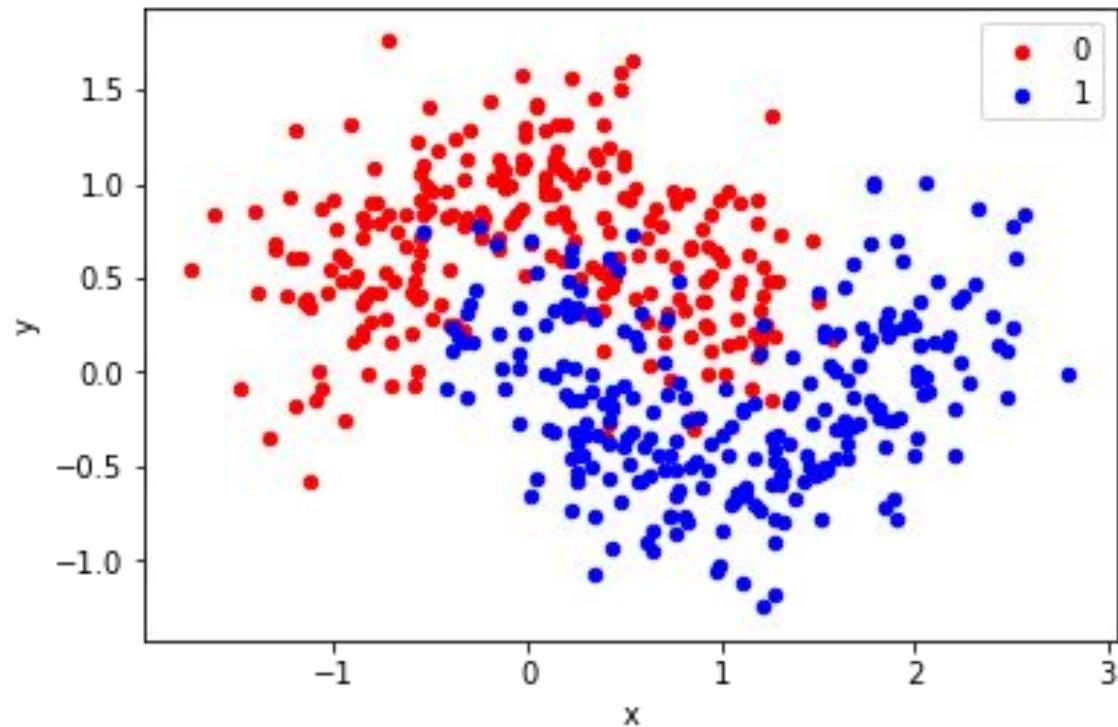
Amount of Say



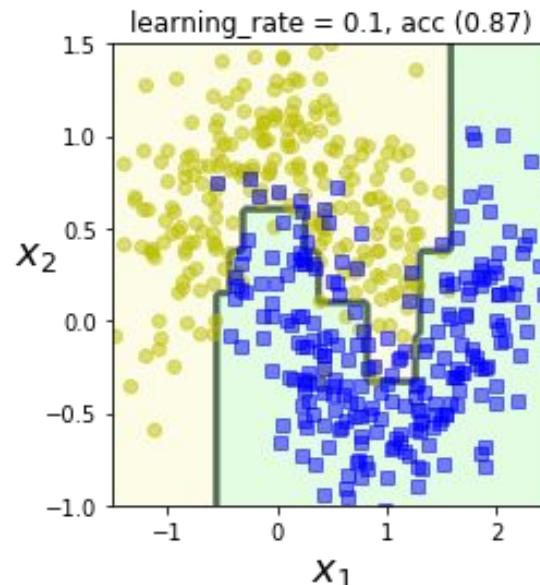
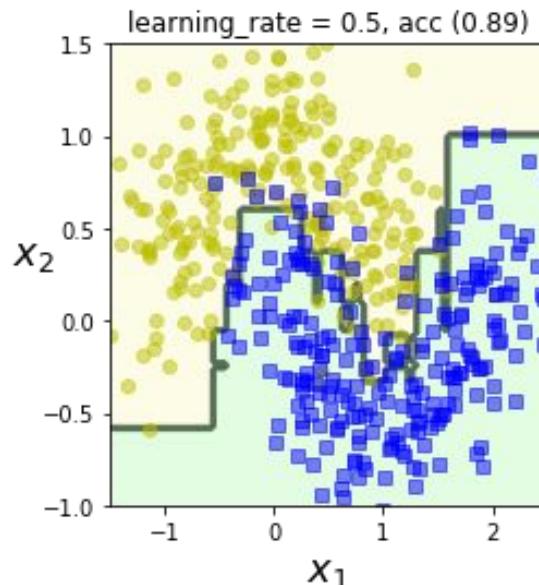
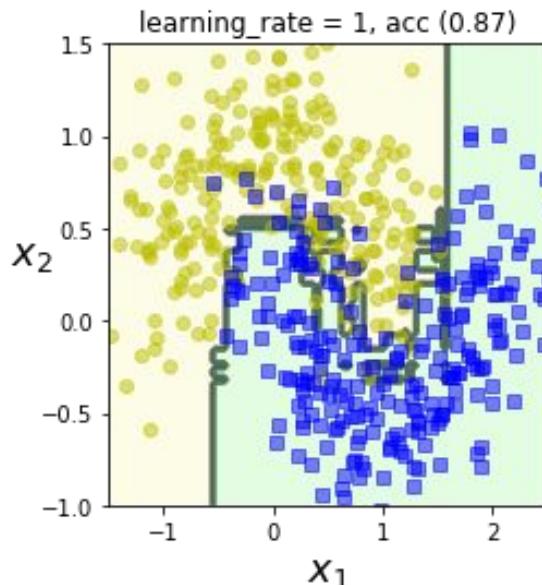
Amount of Say



Case Study



```
ada_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),  
                             n_estimators=200,  
                             algorithm="SAMME.R",  
                             learning_rate=learning_rate,  
                             random_state=42)
```



Gradient Boosting

Greedy Function Approximation: A Gradient Boosting Machine

Jerome H. Friedman*
IMS 1999 Reitz Lecture

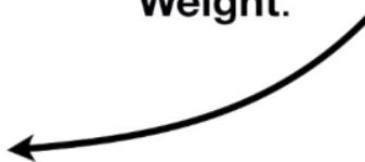
February 24,1999 (modified March 15, 2000, April 19, 2001)

Abstract

Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest-descent minimization. A general gradient-descent “boosting” paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least-absolute-deviation, and Huber-M loss functions for regression, and multi-class logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such “TreeBoost” models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire 1996, and Friedman, Hastie, and Tibshirani 2000 are discussed.

1 Function estimation

...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



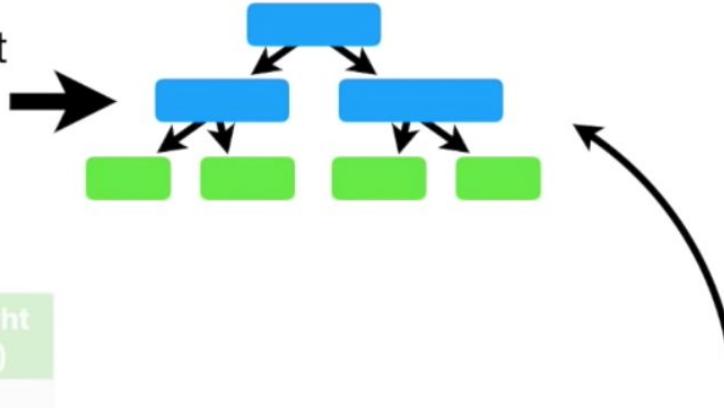
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

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

However, **Gradient Boost** doesn't stop here.

Average Weight

71.2

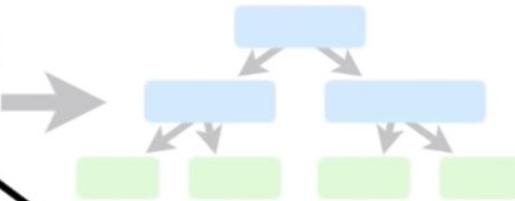


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

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

Average Weight

71.2



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** and the **Predicted Weight, 71.2**.

(Observed Weight - Predicted Weight)

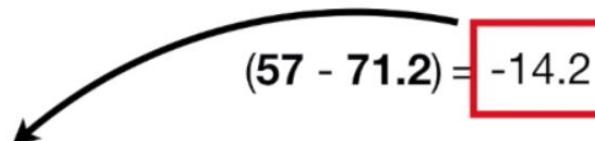
Average Weight

71.2

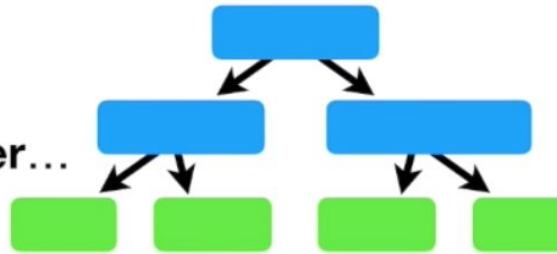
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

Now do the same thing
for the remaining
Weights...

$$(57 - 71.2) = -14.2$$



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

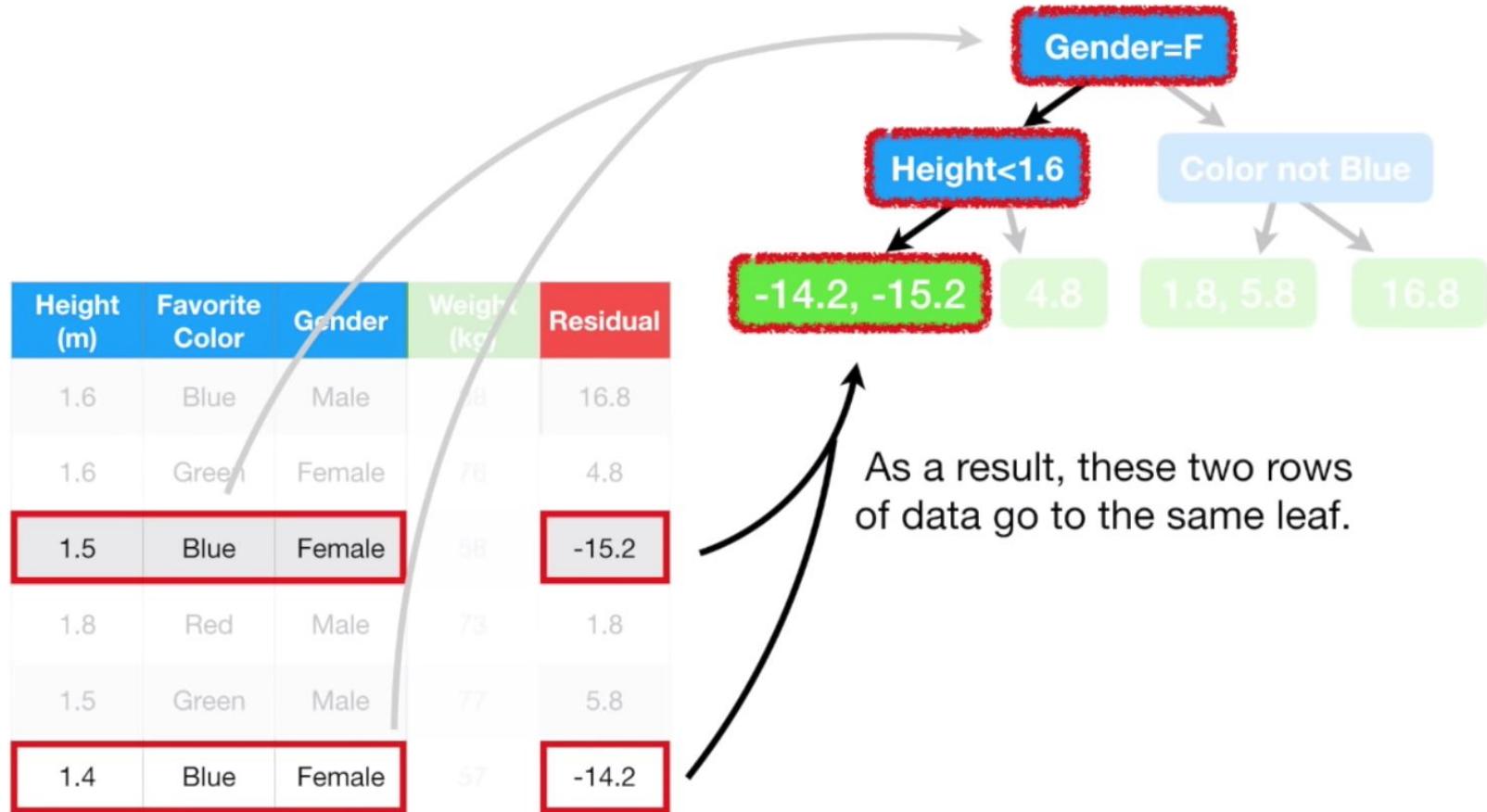
...to **Predict the Residuals.**

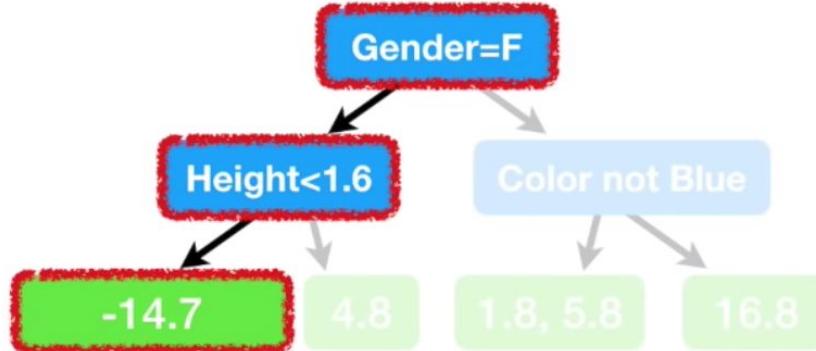
Height (m)	Favorite Color	Gender	Score (0-100)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



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**.





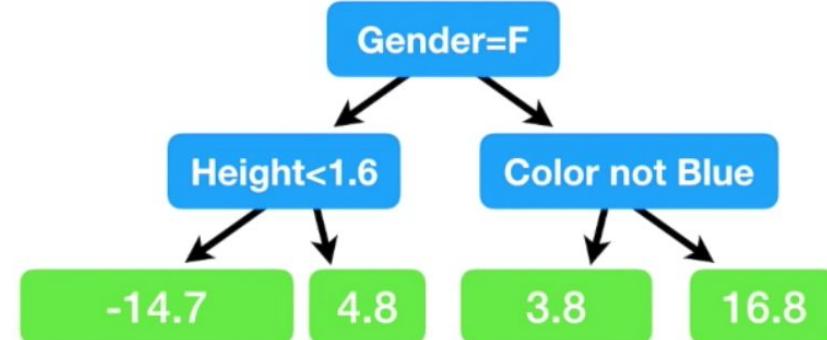
So we replace these residuals with their average.

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

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

Average Weight

71.2



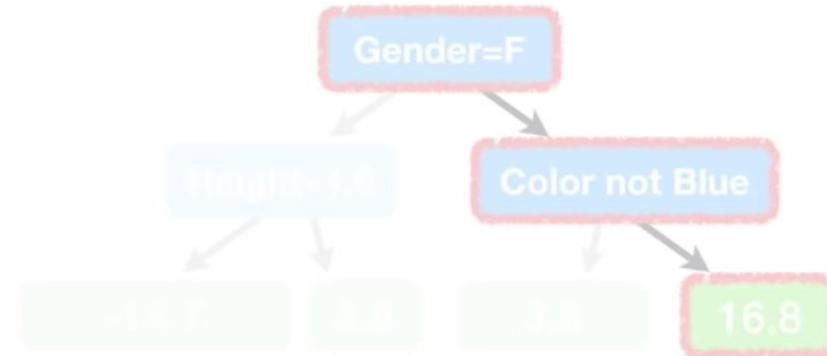
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

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

Average Weight

71.2

+



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

Is this awesome???

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

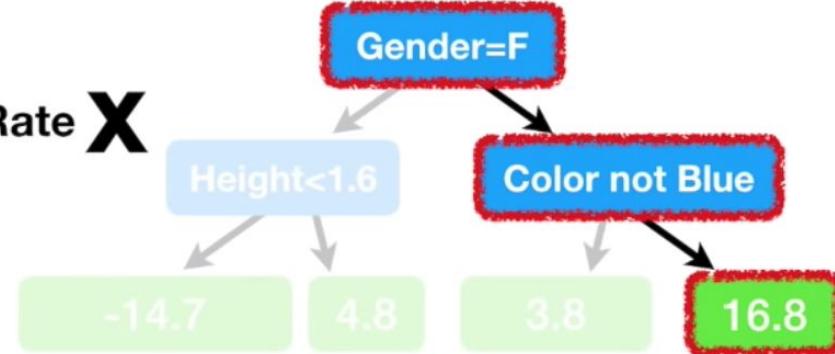
No. The model fits the **Training Data** too well.

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

Average Weight

71.2

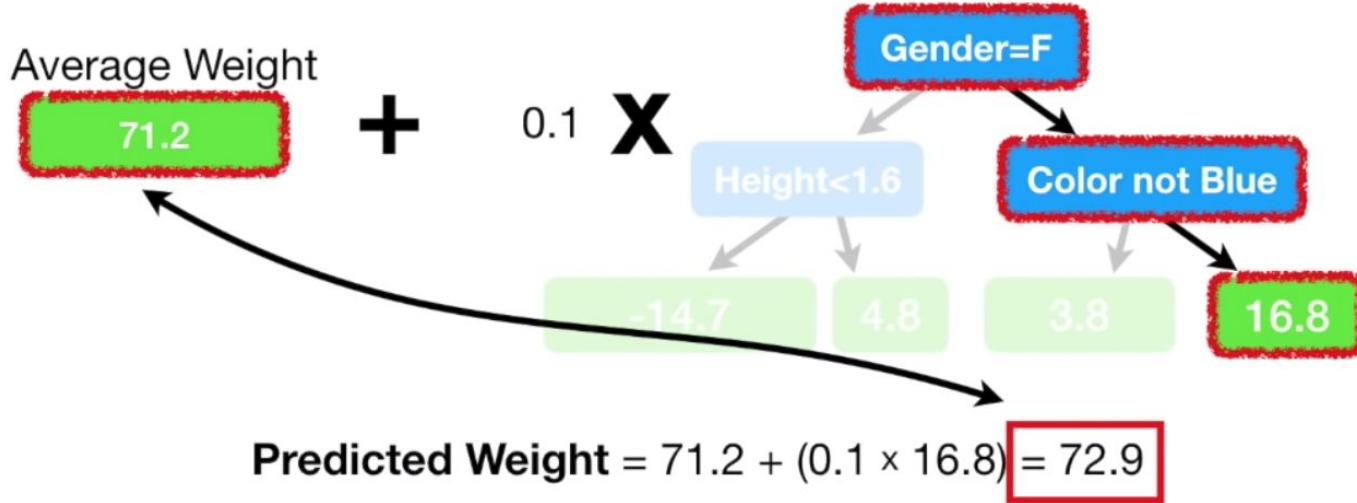
+ Learning Rate \times



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

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

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**.



According to the dude that invented **Gradient Boost**, Jerome Friedman, 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**.



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

Average Weight

71.2

+

0.1

X

Gender=F

Height<1.6

Color not Blue

-14.7

4.8

3.8

16.8

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	
1.4	Blue	Female	57	

$$\text{Residual} = (88 - (71.2 + 0.1 \times 16.8)) \\ = 15.1$$

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

Average Weight

71.2

+

0.1

X

Gender=F

Height<1.6

Color not Blue

-14.7

4.8

3.8

16.8

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

Residual = (Observed - Predicted)

Average Weight

71.2

+

0.1

X

Gender=F

Height<1.6

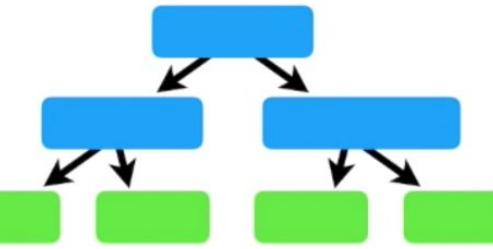
Color not Blue

Residual
16.8
4.8
-15.2
1.8
5.8
-14.2

Residual
15.1
4.3
-13.7
1.4
5.4
-12.7

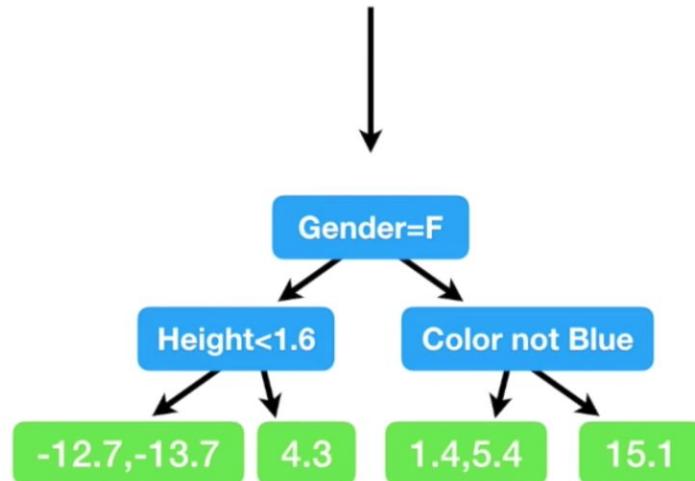
The new **Residuals** are all smaller than before, so we've taken a small step in the right direction.

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



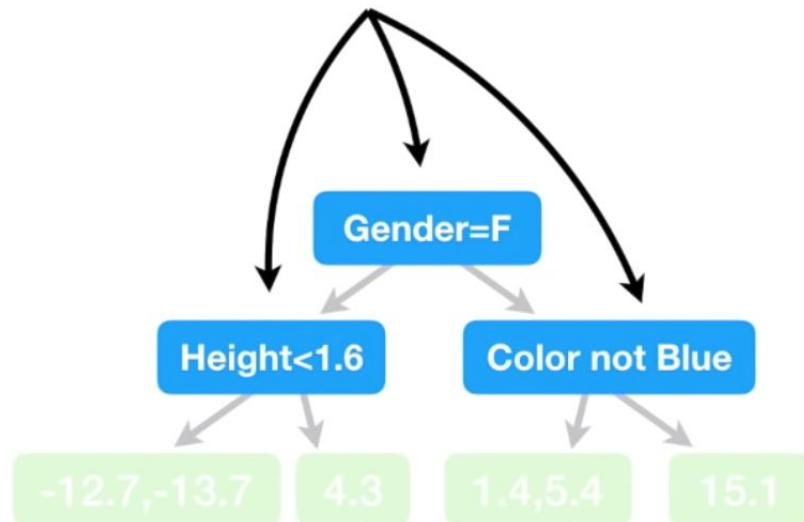
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

And here's the new tree!

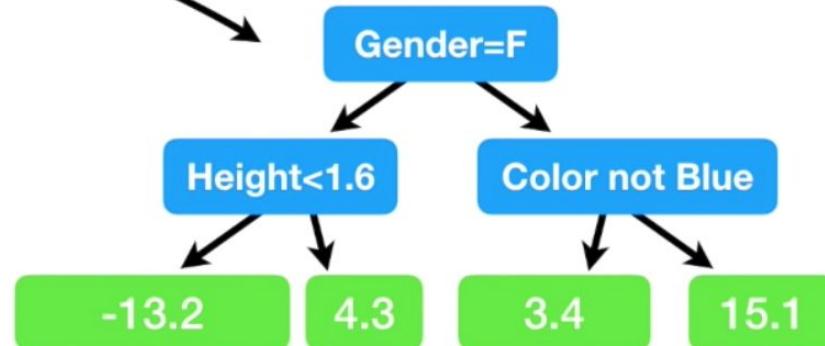


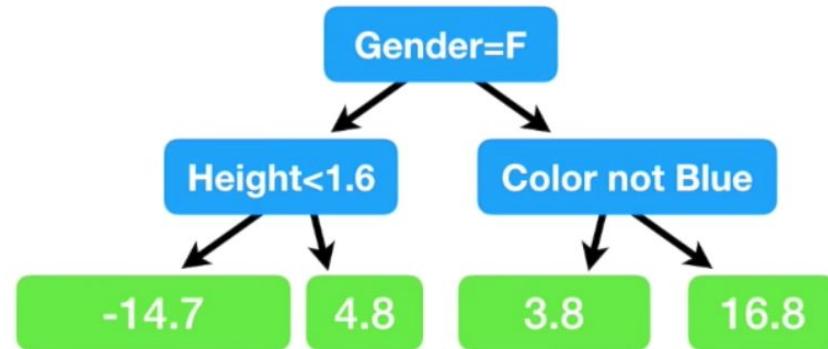
NOTE: In this simple example the branches are the same as before. However, in practice, the trees can be different each time.

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



Now we combine the new **Tree**





Now we combine the new **Tree** with
the previous **Tree**



Average Weight

71.2

Now we combine the new **Tree** with the previous **Tree** and the initial **Leaf**.



Average Weight

71.2

$$+ 0.1 \times$$



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



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

$$+ 0.1 \times$$

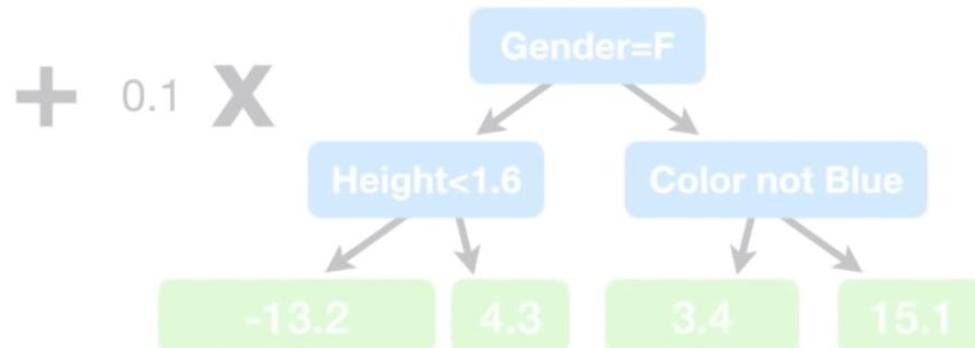


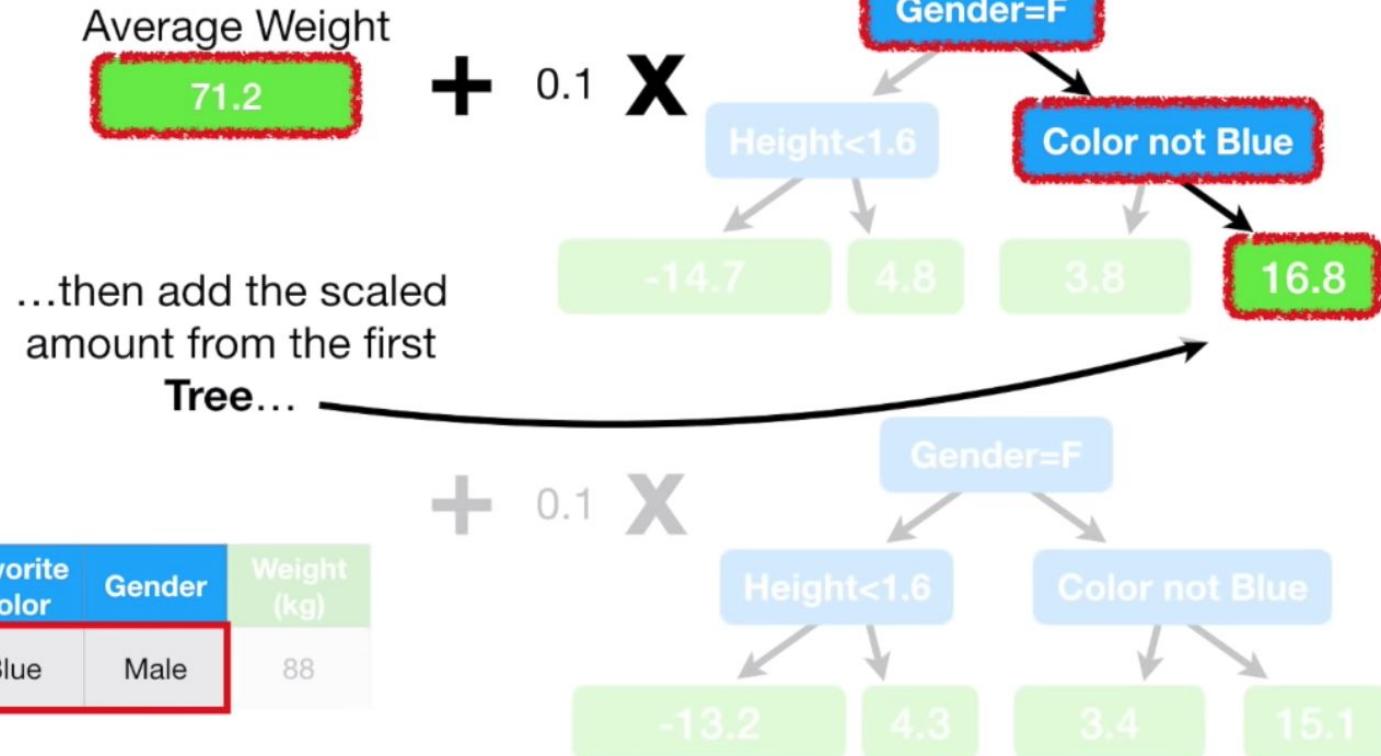
Average Weight
71.2

Just like before, we start with the initial **Prediction...**



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

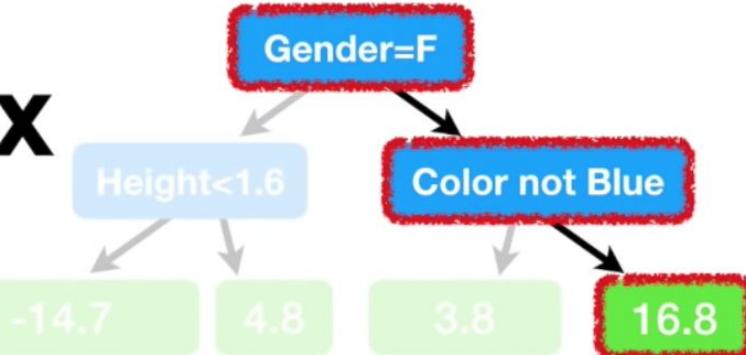




Average Weight

71.2

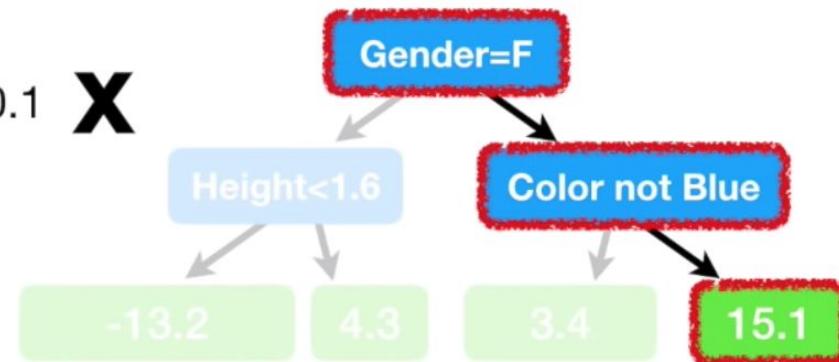
$$+ 0.1 \times$$



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

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

$$+ 0.1 \times$$



Average Weight

71.2

$$+ 0.1 \times$$



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

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

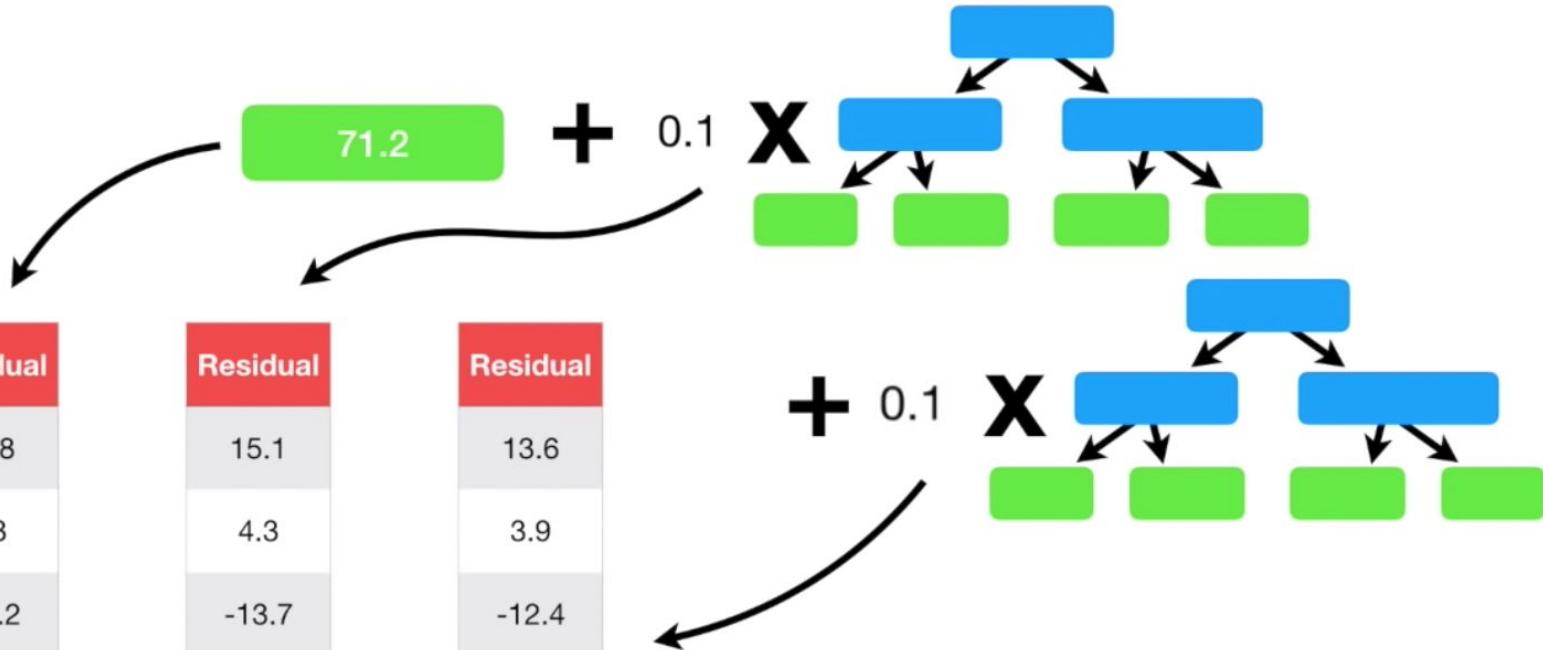
$$= 74.4$$

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

$$+ 0.1 \times$$



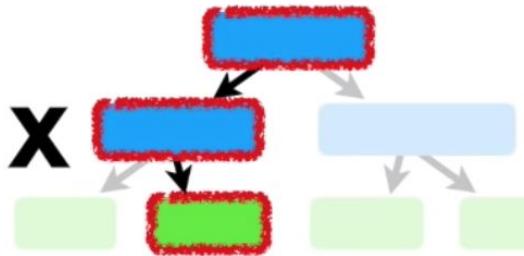
Residual	Residual	Residual
16.8	15.1	13.6
4.8	4.3	3.9
-15.2	-13.7	-12.4
1.8	1.4	1.1
5.8	5.4	5.1
-14.2	-12.7	-11.4



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

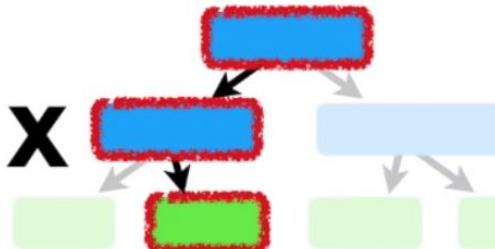
71.2

$+ 0.1 \times$



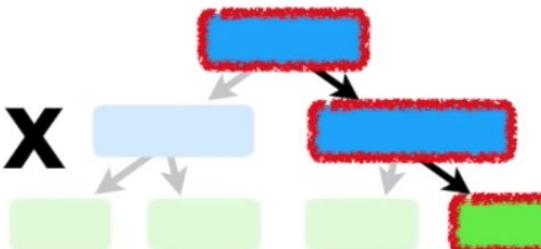
...etc...etc...etc...

$+ 0.1 \times$



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

$+ 0.1 \times$



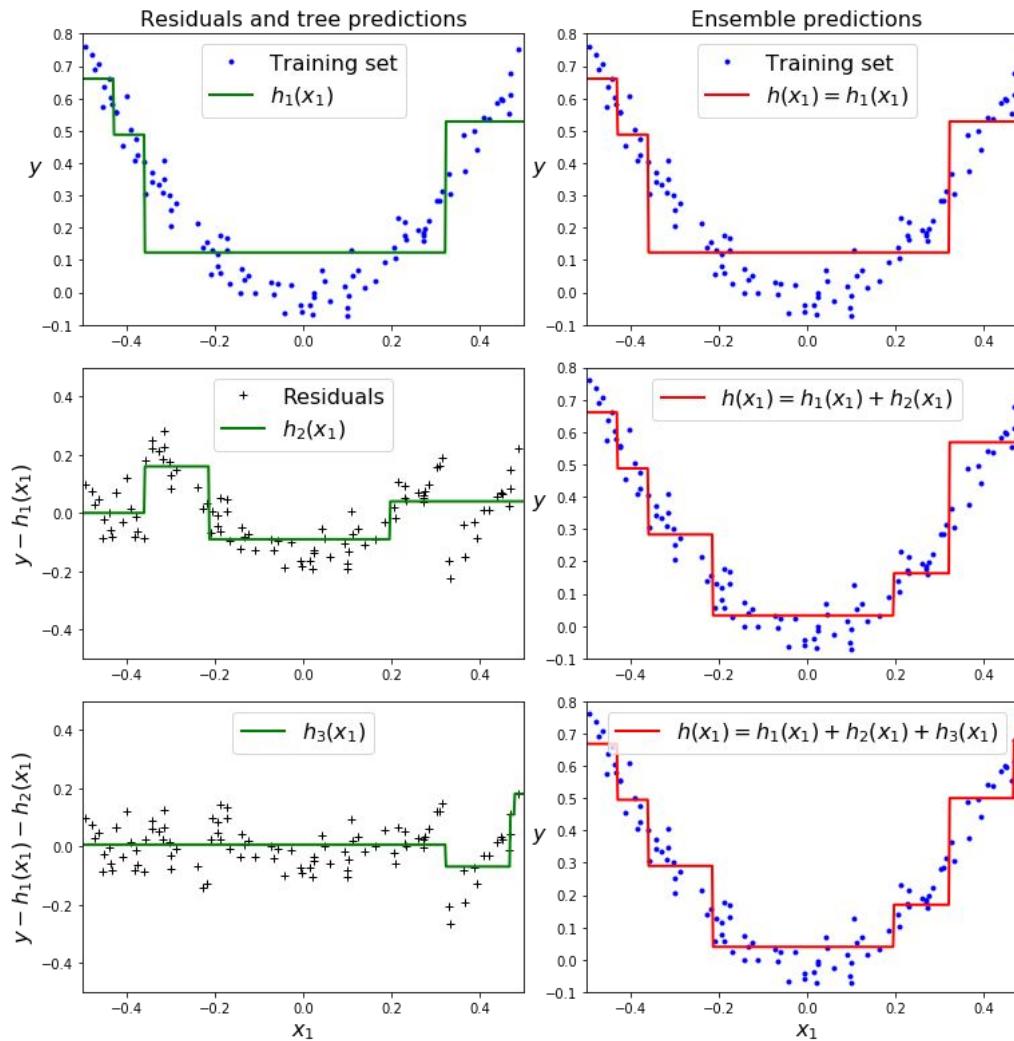
...etc...etc...etc...

```
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

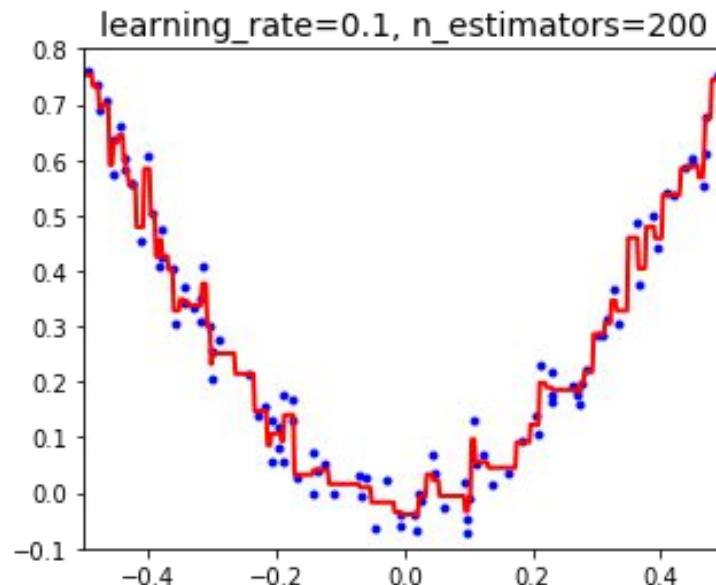
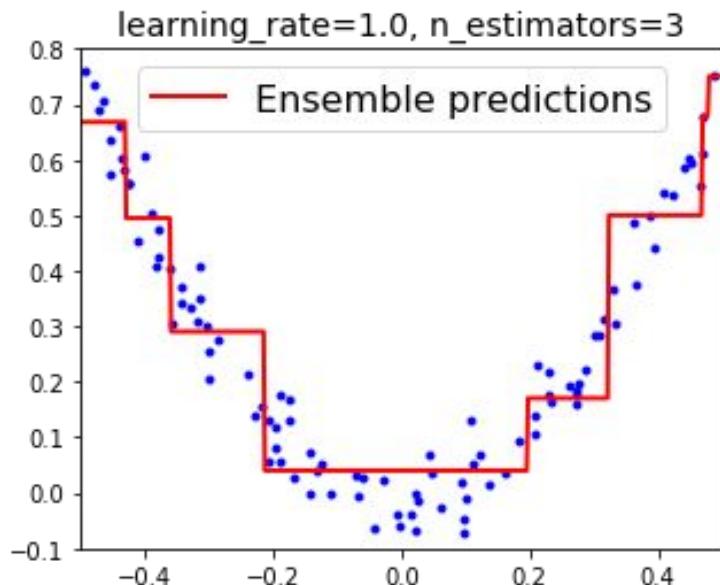
# create a simple DecisionTreeRegressor
tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)

# now train a second DecisionTreeRegressor
# on the residual errors made by the first predictor
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)

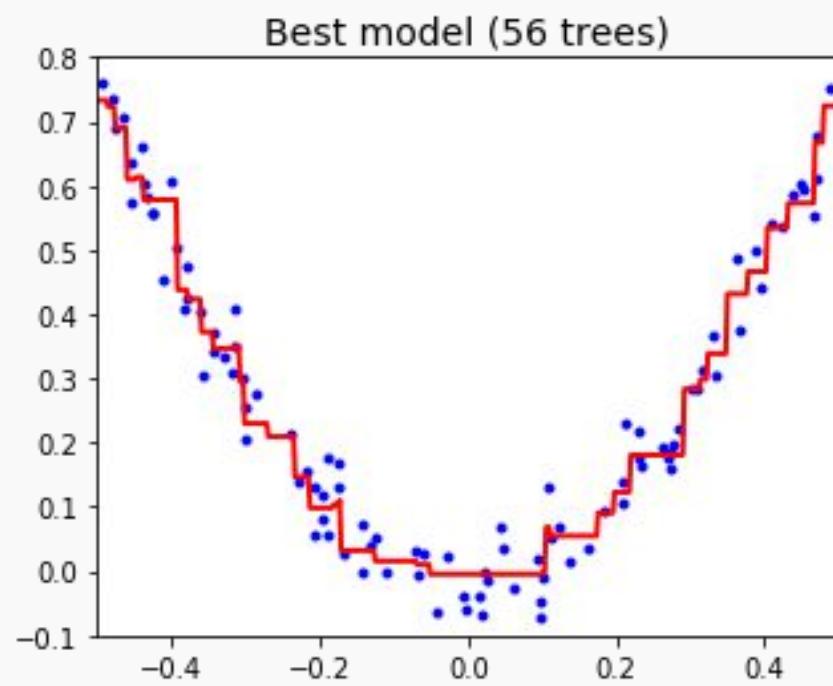
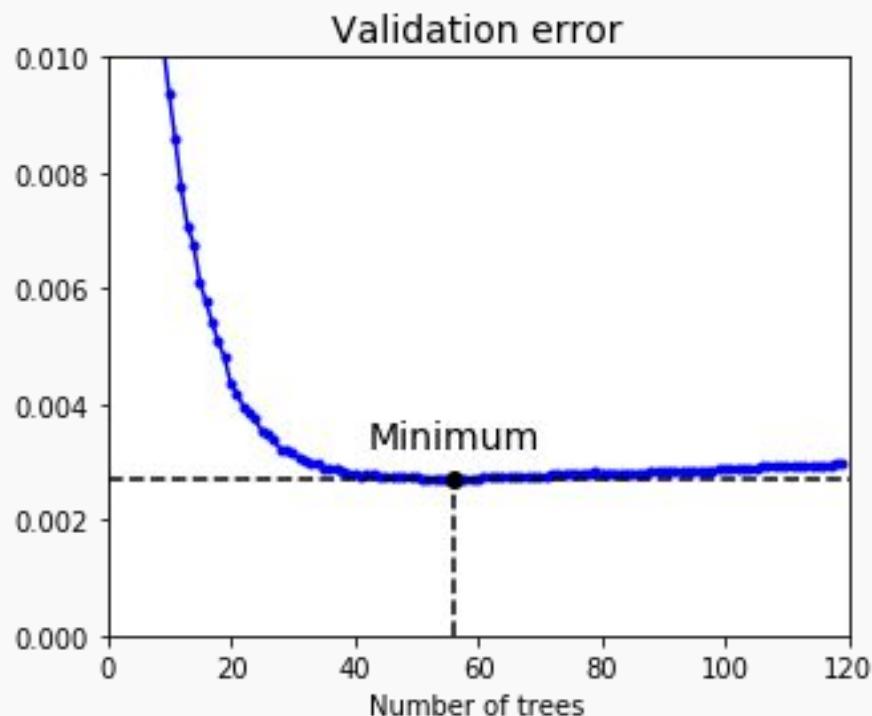
# then we train a third regressor on the residual errors
# made by the second predictor
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```



```
gbrt = GradientBoostingRegressor(max_depth=2,  
                                 n_estimators=3,  
                                 learning_rate=1.0,  
                                 random_state=42)
```



Early Stopping



Improvements to Basic Gradient Boosting (best practices)

1. Tree Constraints

- a. the advice is to keep adding trees until no further improvement is observed.
- b. tree depth, deeper trees are more complex trees and shorter trees are preferred.
- c. generally, better results are seen with 4-8 levels.

2. Shrinkage

- a. decreasing the value of the learning rate increases the best value for the number of trees.
- b. it is common to have small values for learning rate in the range of 0.1 to 0.3, as well as values less than 0.1.

3. Random Sampling (Stochastic Gradient Boosting)

- a. at each iteration a subsample of the training data is drawn at random
- b. generally, aggressive sub-sampling such as selecting only 50% of the data has shown to be beneficial.

4. Penalized Gradient Boosting (L1, L2)

dmlc
XGBoost

<https://xgboost.ai>



XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

ABSTRACT

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, we provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems.

Keywords

Large-scale Machine Learning

1. INTRODUCTION

Machine learning and data-driven approaches are becoming very important in many areas. Smart spam classifiers protect our email by learning from massive amounts of spam data and user feedback; advertising systems learn to match the right ads with the right context; fraud detection systems protect banks from malicious attackers; anomaly event detection systems help experimental physicists to find events that lead to new physics. There are two important factors that drive these successful applications: usage of effective

problems. Besides being used as a stand-alone predictor, it is also incorporated into real-world production pipelines for ad click through rate prediction [15]. Finally, it is the de-facto choice of ensemble method and is used in challenges such as the Netflix prize [3].

In this paper, we describe XGBoost, a scalable machine learning system for tree boosting. The system is available as an open source package². The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions³ published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. For comparison, the second most popular method, deep neural nets, was used in 11 solutions. The success of the system was also witnessed in KDDCup 2015, where XGBoost was used by every winning team in the top-10. Moreover, the winning teams reported that ensemble methods outperform a well-configured XGBoost by only a small amount [1].

These results demonstrate that our system gives state-of-the-art results on a wide range of problems. Examples of the problems in these winning solutions include: store sales prediction; high energy physics event classification; web text classification; customer behavior prediction; motion detection; ad click through rate prediction; malware classification;

Why use XGBoost?

- Execution Speed
- Model Performance
- Good Documentation

A wide-angle photograph of a vast mountain range, likely the Himalayas, under a clear blue sky. The mountains are heavily covered in white snow, with deep shadows in the valleys. In the foreground, there are patches of green vegetation and rocky terrain partially obscured by low-hanging clouds.

XGBoost

Gradient Boosting > Random Forest > Bagging > Single Trees



Search or jump to...



Pull requests Issues Marketplace Explore

<https://github.com/szilard/GBM-perf>[szilard / GBM-perf](#)[Watch ▾ 17](#)[Unstar](#)

156

[Fork](#)

20

[Code](#)[Issues 27](#)[Pull requests](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[Insights](#)[master ▾](#)[1 branch](#)[0 tags](#)[Go to file](#)[Add file ▾](#)[Code ▾](#)

About

Performance of various open source
GBM implementations

[machine-learning](#)[gradient-boosting-machine](#)[gbm](#)[h2oai](#) [xgboost](#) [lightgbm](#)[benchmark](#)[Readme](#)[MIT License](#)

Releases

No releases published

	szilard random forests	0c75dcd 16 days ago	399 commits
	analysis random forests	16 days ago	
	cpu switch lightgbm from OHE tp cats in the main benchmark	last month	
	gpu switch lightgbm from OHE tp cats in the main benchmark	last month	
	v1-keep-old-before_docker archive v1 before docker	3 years ago	
	wip-testing h2o super-linear	29 days ago	
	.gitignore lightgbm GPU	3 years ago	
	GBM-perf.Rproj Rstudio	3 years ago	
	LICENSE add licence (MIT)	2 years ago	
	README.md docs	20 days ago	

CPU

r4.8xlarge (32 cores, but run on physical cores only/no hyperthreading) with software as of 2020-09-08:

Tool	Time[s] 100K	Time[s] 1M	Time[s] 10M	AUC 1M	AUC 10M
h2o	14	16	90	0.762	0.776
xgboost	0.8	5.5	70	0.748	0.754
lightgbm	2.1	4.0	20	0.765	0.792
catboost	3.8	10	80	0.734	0.735

GPU

p3.2xlarge (1 GPU, Tesla V100) with software as of 2020-09-08 (and CUDA 11.0):

Tool	Time[s] 100K	Time[s] 1M	Time[s] 10M	AUC 1M	AUC 10M
h2o xgboost	6.4	14	55	0.749	0.756
xgboost	3.4	6.4	11	0.748	0.754
lightgbm	8	12	50	0.763	0.792
catboost	1.8	4.7	36	0.732 ?!	0.736 ?!

Recommendations

If you **don't have a GPU**, **lightgbm** (CPU) trains often the fastest (on larger data).

If you **have a GPU**, **xgboost** (GPU) is very fast (and depending on the data, your hardware etc. often faster than the above mentioned lightgbm on CPU).

If you consider deployment, **h2o** has the best ways to deploy as a real-time (fast scoring) application.

Note, however, there are a lot more other criteria to consider when you choose which tool to use, e.g.:

	gbm (R pkg)	xgboost	lightgbm	h2o
easy R install	cran	cran	linux OK	java+cran
maintained	retired	yes	yes	yes
preprocessing	not needed	1-hot	1-hot/categ int	not needed
new cats scoring	yes	no	no	yes
early stopping	no	yes	yes	yes
speed (CPU)	1 core	ok	fastest	slow (small data)
GPU supported	no	yes	yes	via xgboost
speed GPU	NA	fastest	ok/slow	indirectly/slower
REST scoring	no	no	no	yes
other algos	no	RF	RF	RF/GLM/NN
best for	teaching/historic	Kaggle	Kaggle	prod/real-time

Recommendation Hyperparameter Tuning

```
search_space = [{"clf": [XGBClassifier()],
                 "clf__n_estimators": [100, 200, 300, 400],
                 "clf__max_depth": [3, 6, 8, 10, 15],
                 "clf__learning_rate": [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                 "clf__subsample": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0],
                 "clf__colsample_bytree": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0],
                 "clf__tree_method": ["auto", "gpu_hist"]}]
```

~20k combinations 

Ensemble Learning case study: Running XGBoost on Google Colab free GPU



Gabriel Signoretti Oct 22, 2019 · 11 min read ★

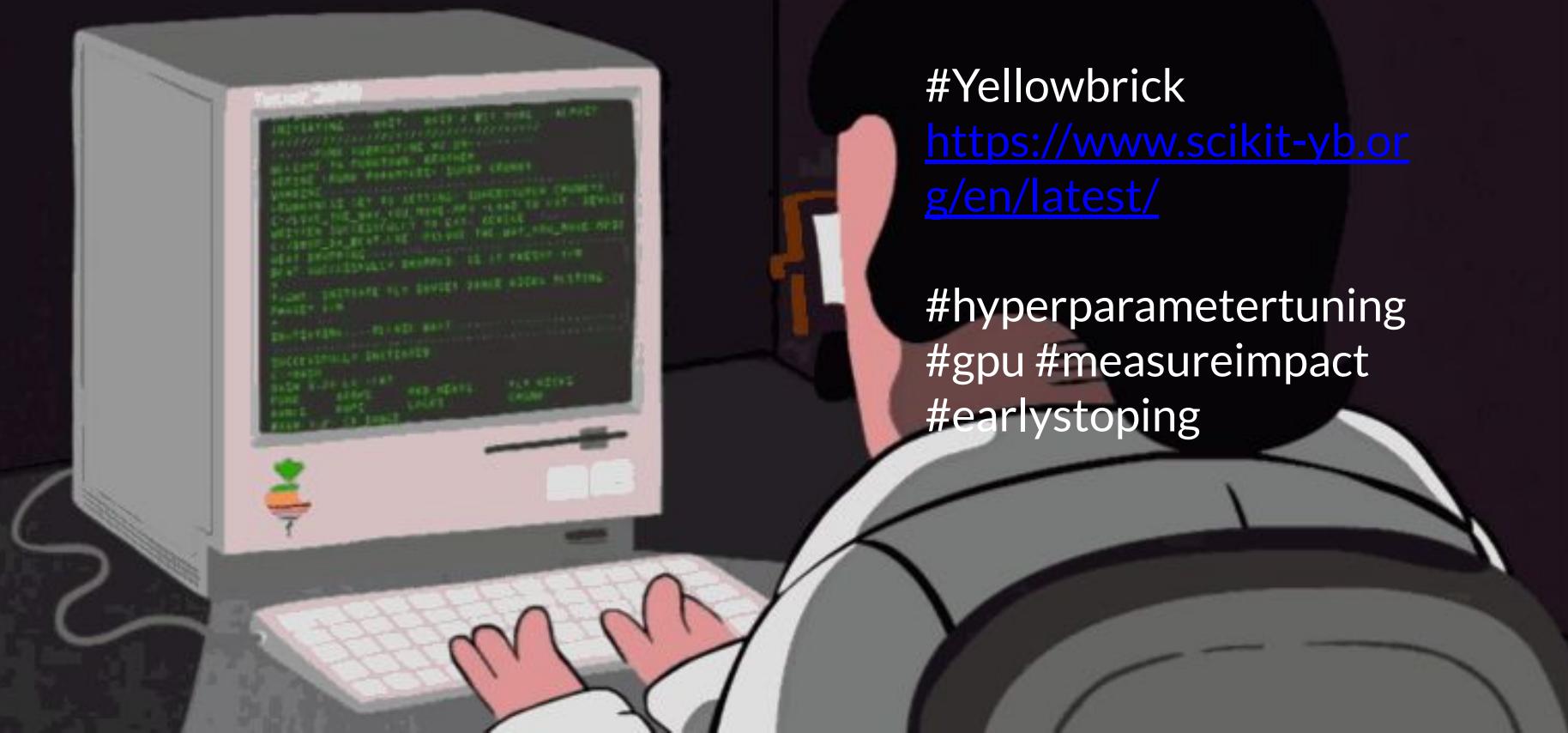


...

This article is the second part of a case study where we are exploring the 1994 census income dataset. In the [first part](#), we took a deeper look at the dataset, compared the performance of some ensemble methods and then explored some tools to help with the model interpretability.

In this second part, we will explore a technique called **Gradient Boosting** and the **Google Colaboratory**, which is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

Lesson #07 - Ensemble Learning II.ipynb



#Yellowbrick

[https://www.scikit-yb.org
en/latest/](https://www.scikit-yb.org/en/latest/)

#hyperparametertuning

#gpu #measureimpact

#earlystoping