Data Analytics for Digital Health
Ensemble Methods

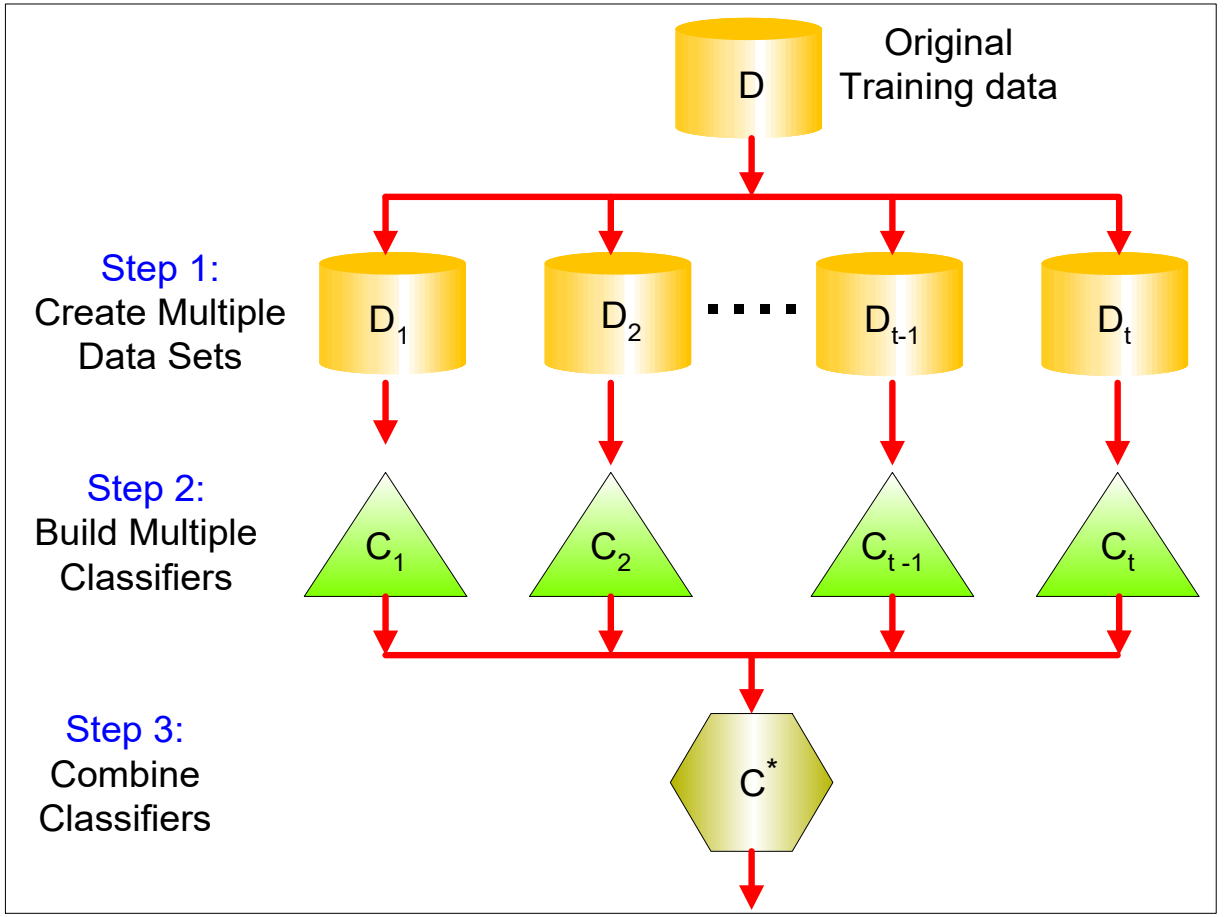
Riccardo Guidotti, Francesca Naretto

Slides edited from Tan, Steinbach, Kumar, Introduction to Data Mining

Università di Pisa

Ensemble Methods

- Improves the accuracy by **aggregating the predictions** of multiple classifiers.
- Construct a set of **base classifiers** from the training data.
- Predict class label of test records by combining the predictions made by multiple classifiers.

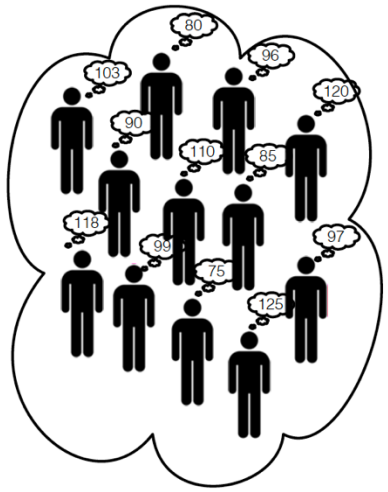


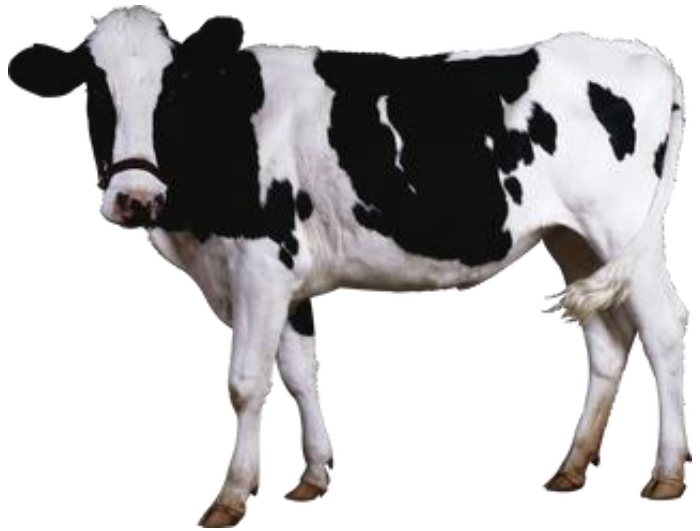
The Wisdom of the Crowds

The Wisdom of the Crowds

- The **collective knowledge** of a diverse and independent body of people typically **exceeds** the knowledge of any **single individual** and can be harnessed by voting.

- How much does the cow weigh?
- Crowd's prediction is the average of all predictions.
- This crowd predicts 99.8333 Kg
- The cow weighs 99.657 Kg.



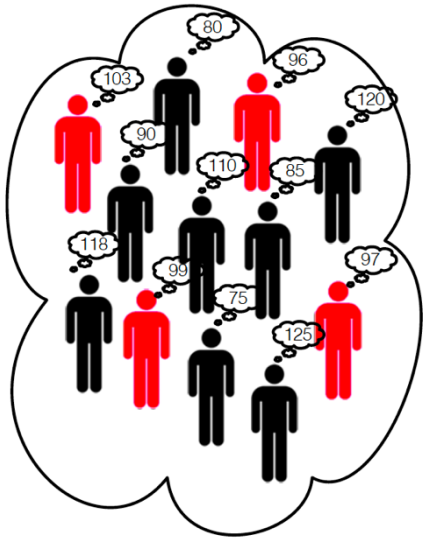


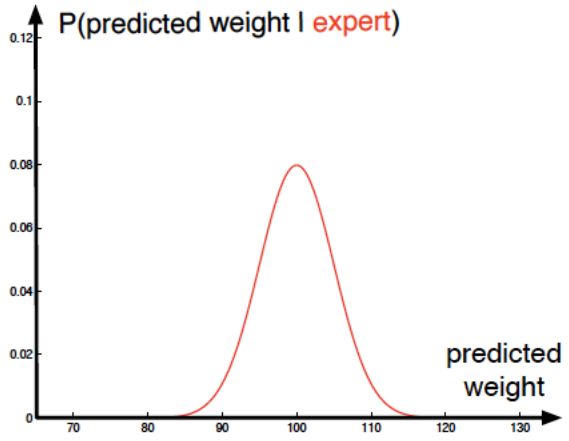
Has crowd made a good estimate?

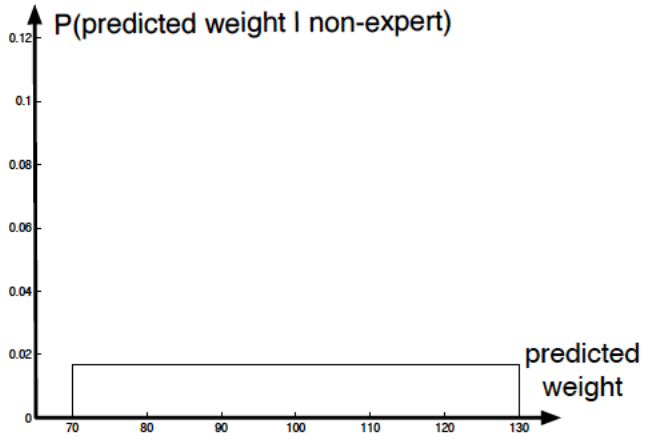
If composition of crowd:

- 30% experts.
- 70% non-experts.

and their level of expertise:







$P(\text{pred. weight} \mid \text{expert}) :$

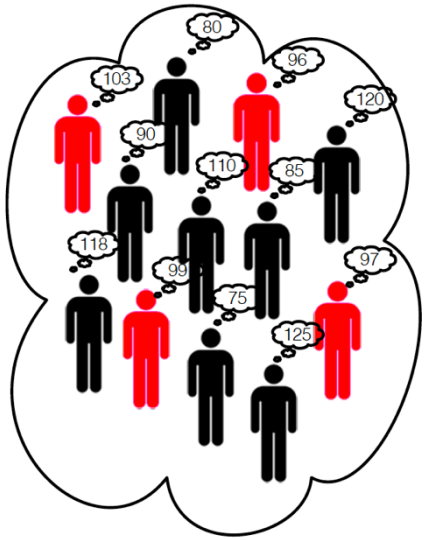
$\mathcal{N}(100, 5^2)$

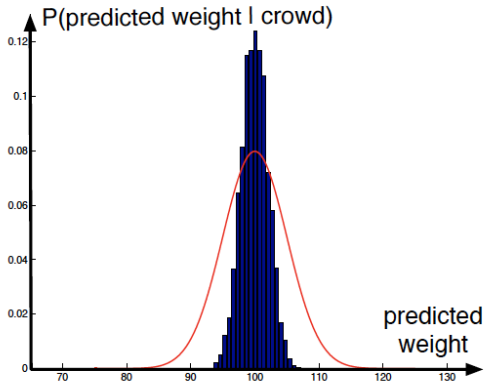
$P(\text{pred. weight} \mid \text{non-expert}) :$

$\mathcal{U}(70, 130)$

Has crowd made a good estimate?

- If the crowd contains 50 independent people:
- On average this crowd will make better estimates than the experts.
- Thus the crowd is wiser than each of the experts!





$\Uparrow$

$P(\text{pred. weight}|\text{crowd})$ **and**
$P(\text{pred. weight}|\text{expert})$

Why didn’t I just asked a bunch of experts?

- Large enough crowd → high probability a sufficient number of experts will be in crowd (for any question).
- Random selection → don’t make a biased choice in experts.
- For some questions it may be hard to identify a diverse set of experts.

What makes a crowd wise?

- According to James Surowiecki there are four elements required to form a wise crowd
- **Diversity of opinion**. People in crowd should have a range of experiences, education and opinions (this encourages independent predictions).
- **Independence**. Prediction by person in crowd is not influenced by other people in the crowd.
- **Decentralization**. People have specializations and local knowledge.
- **Aggregation**. There is a mechanism for aggregating all predictions into one single prediction.

Back to Machine Learning

Back to Machine Learning

It will exploit *Wisdom of crowd* ideas for specific tasks

- By combining classifier predictions and aims to combine independent and diverse classifiers.

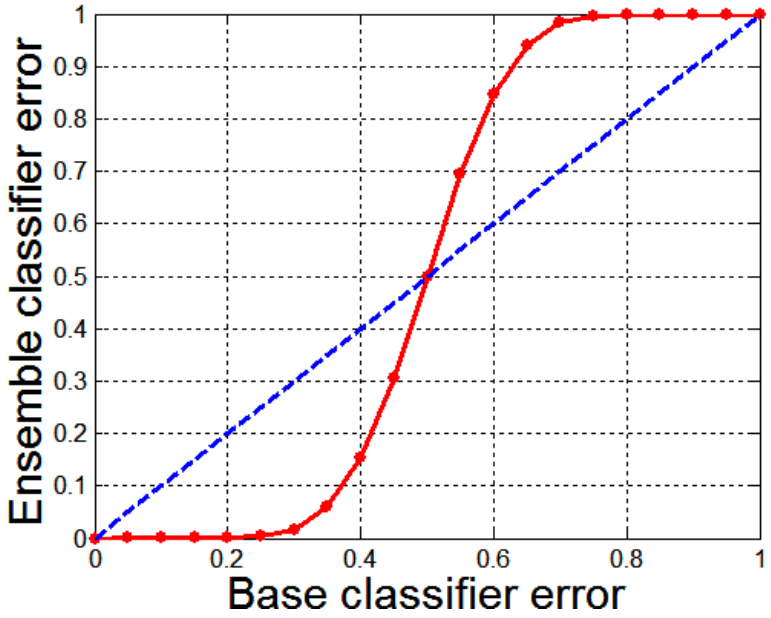
But it will use labelled training data

- to identify the **expert** classifiers in the pool;
- to identify **complementary** classifiers;
- to indicate how to the best **combine** them.

Why Ensemble Methods work?

Suppose there are 25 base classifiers

- Each classifier has error rate, $\varepsilon = 0.35$
- Assume errors made by classifiers are uncorrelated
- Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Types of Ensemble Methods

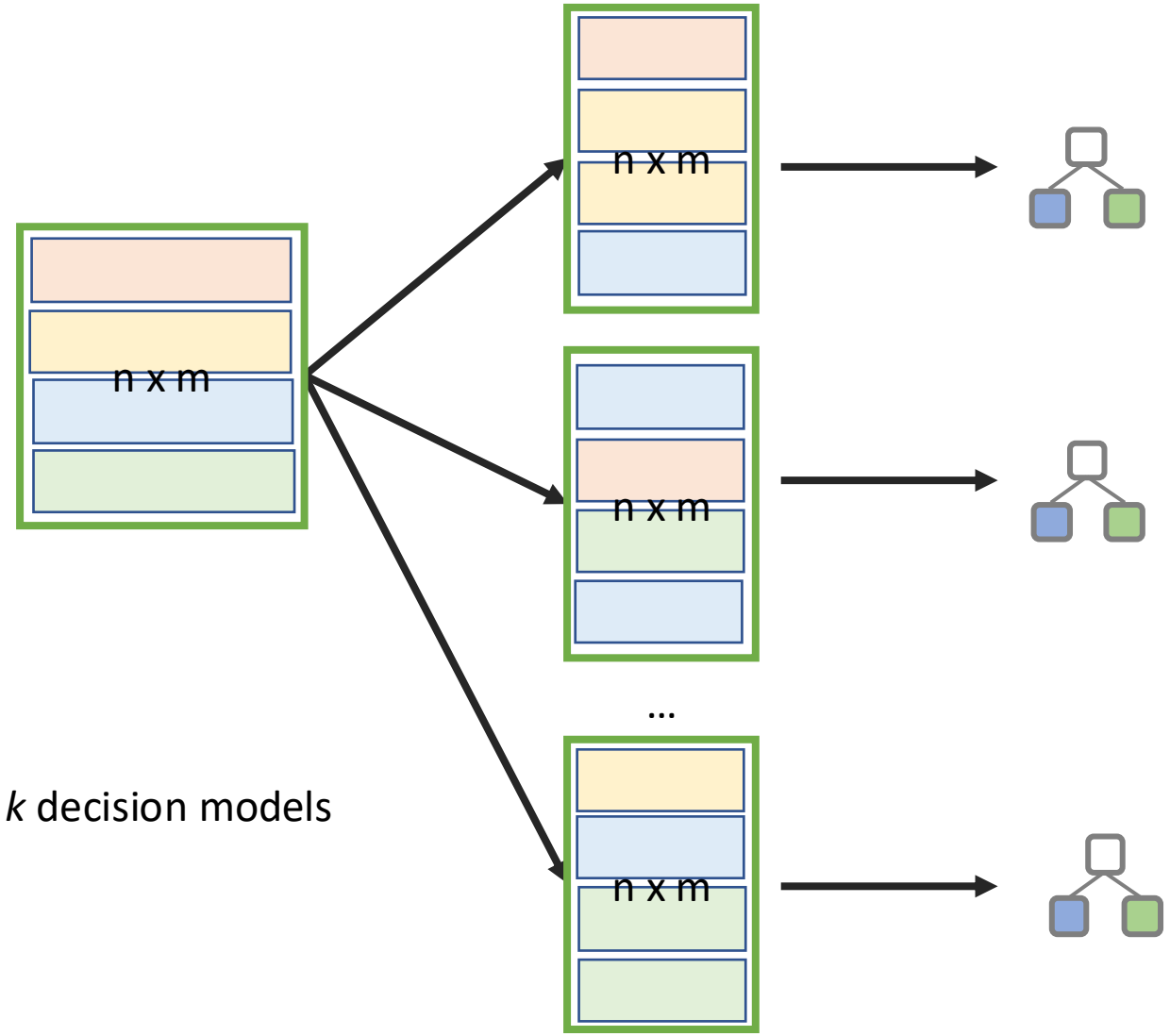
- Manipulate data distribution
  - Example: bagging, boosting
- Manipulate input features
  - Example: random forests
- Manipulate class labels
  - Example: error-correcting output coding

Bagging

Bootstrap

- It is a re-sampling statistical technique with re-entry to approximate the sample distribution of a statistic.
- We consider a dataset with n records $X = \{x_1, \ldots, x_n\}$.
- From X we sample m datasets of constant size n, say $\{X_1^*, \ldots, X_m^*\}$.
- In each bootstrap extraction X_i^*, each record in the original dataset has $1/n$ probability of being extracted and can be extracted more than once or zero.

Bagging (a.k.a. Bootstrap AGGregatING)



Bagging

- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample
- Each sample has probability *1/n* of being selected

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

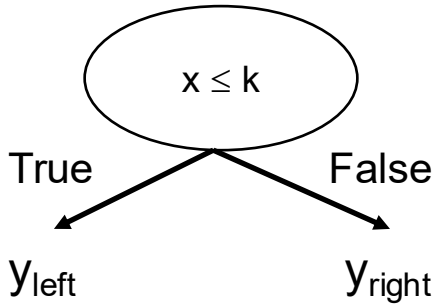
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x <= k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

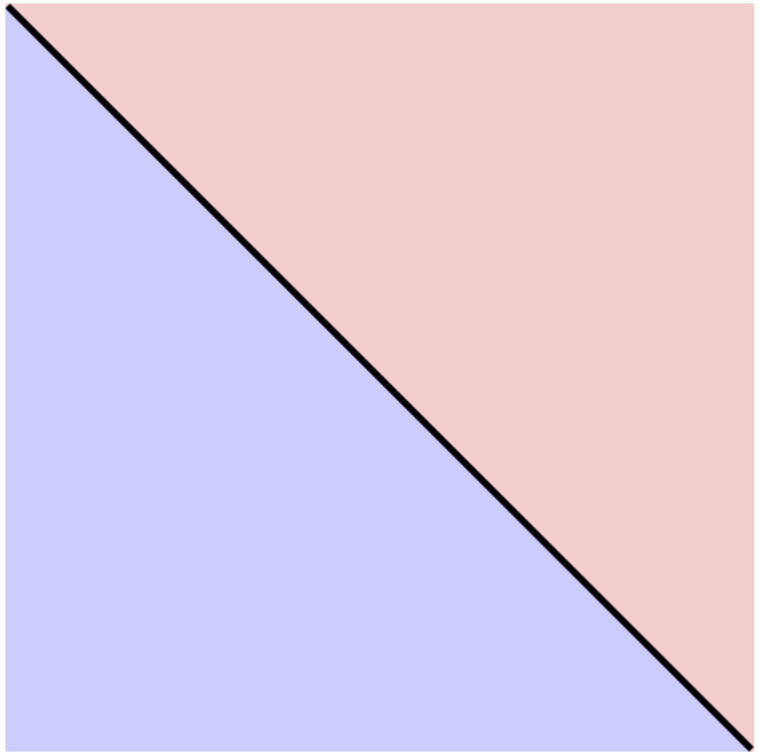
Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

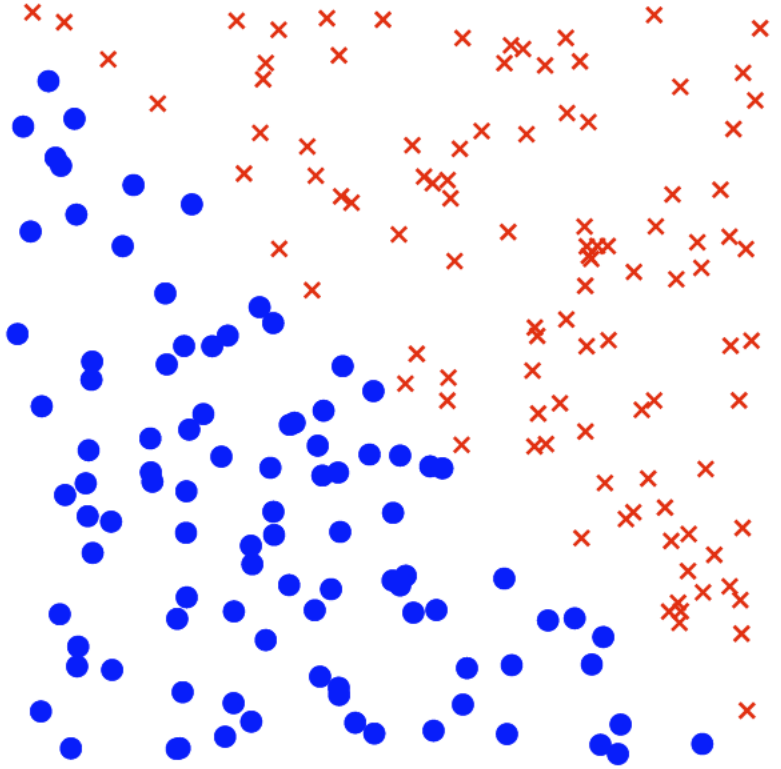
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging Visual Example

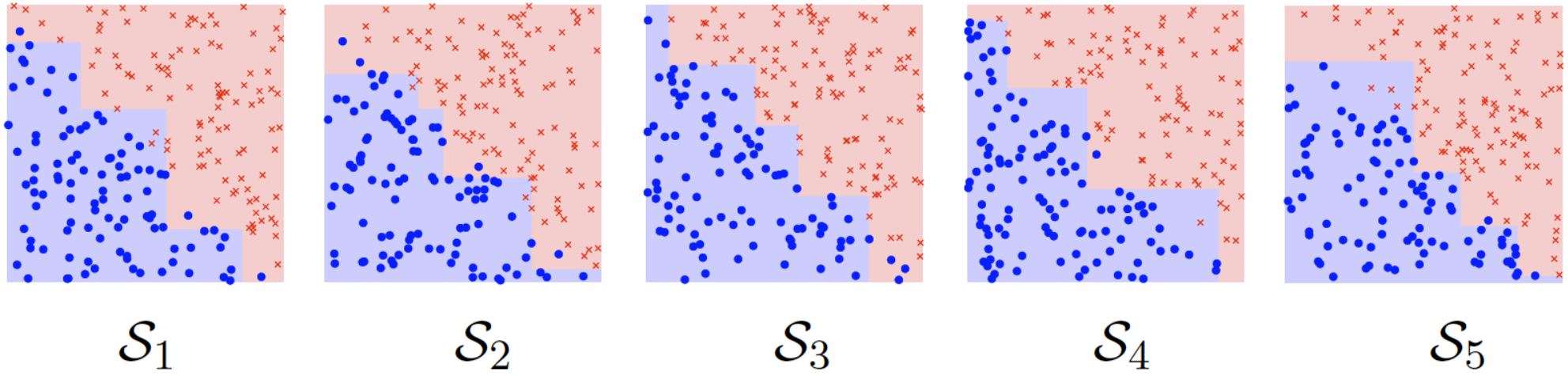


True decision boundary

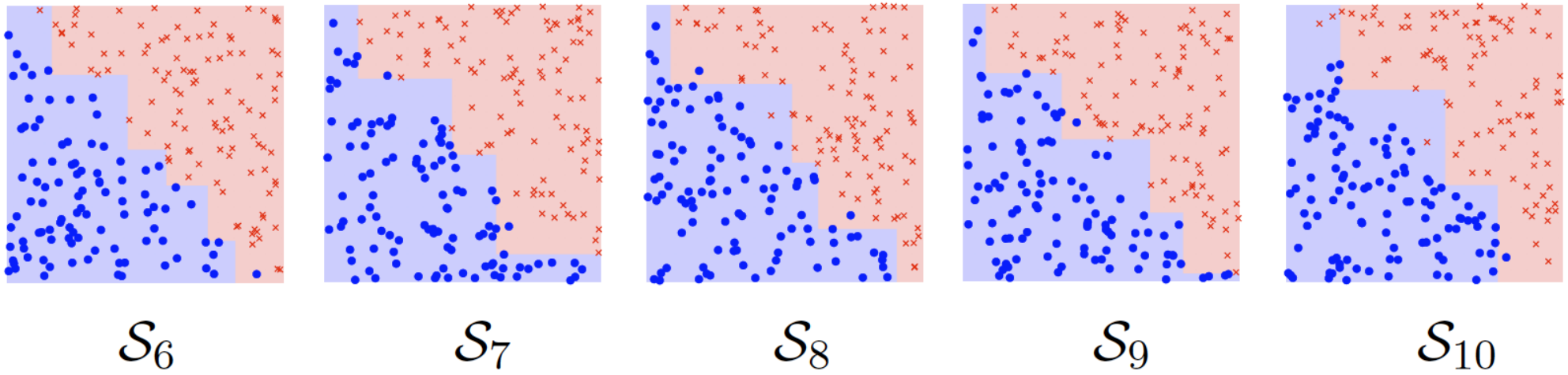


Training data

Bagging Visual Example



$\mathcal{S}_1$ $\mathcal{S}_2$ $\mathcal{S}_3$ $\mathcal{S}_4$ $\mathcal{S}_5$

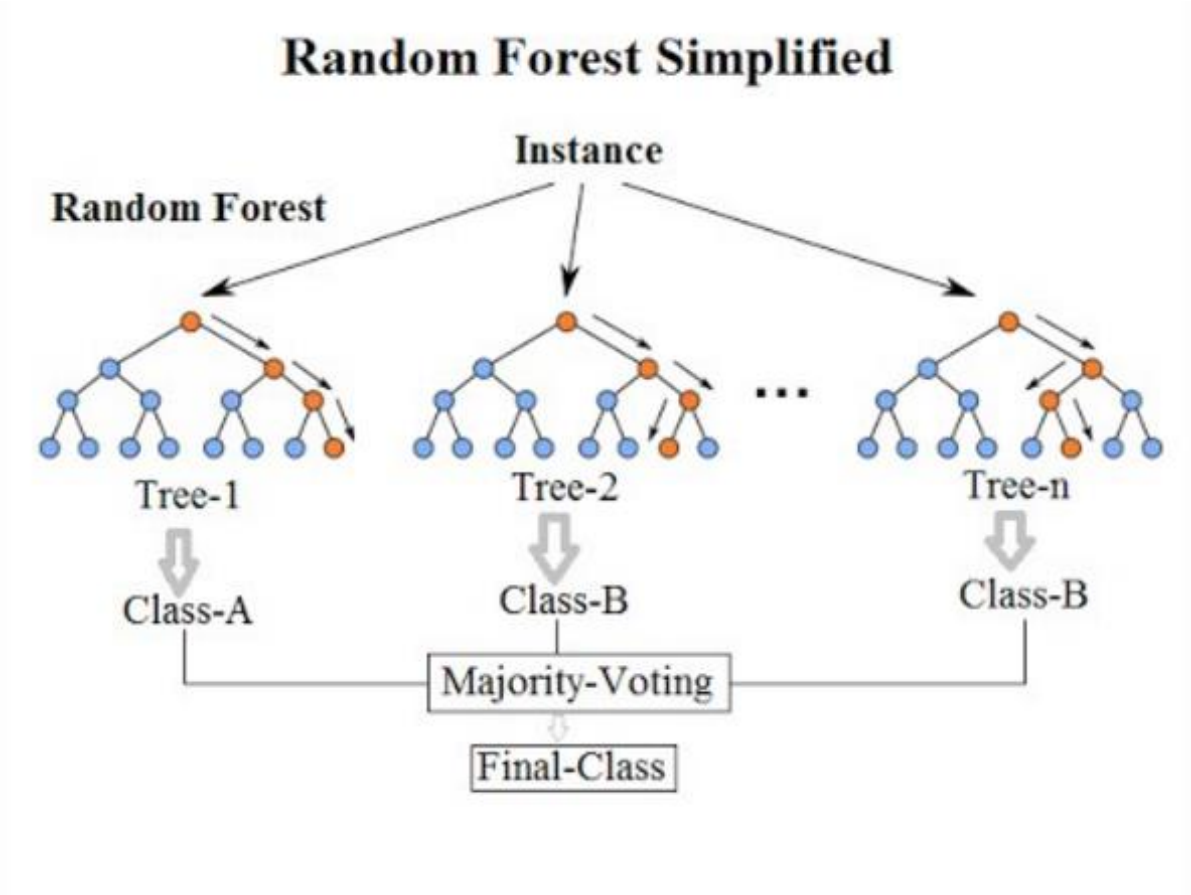


$\mathcal{S}_6$ $\mathcal{S}_7$ $\mathcal{S}_8$ $\mathcal{S}_9$ $\mathcal{S}_{10}$

Random Forests

Random Forests

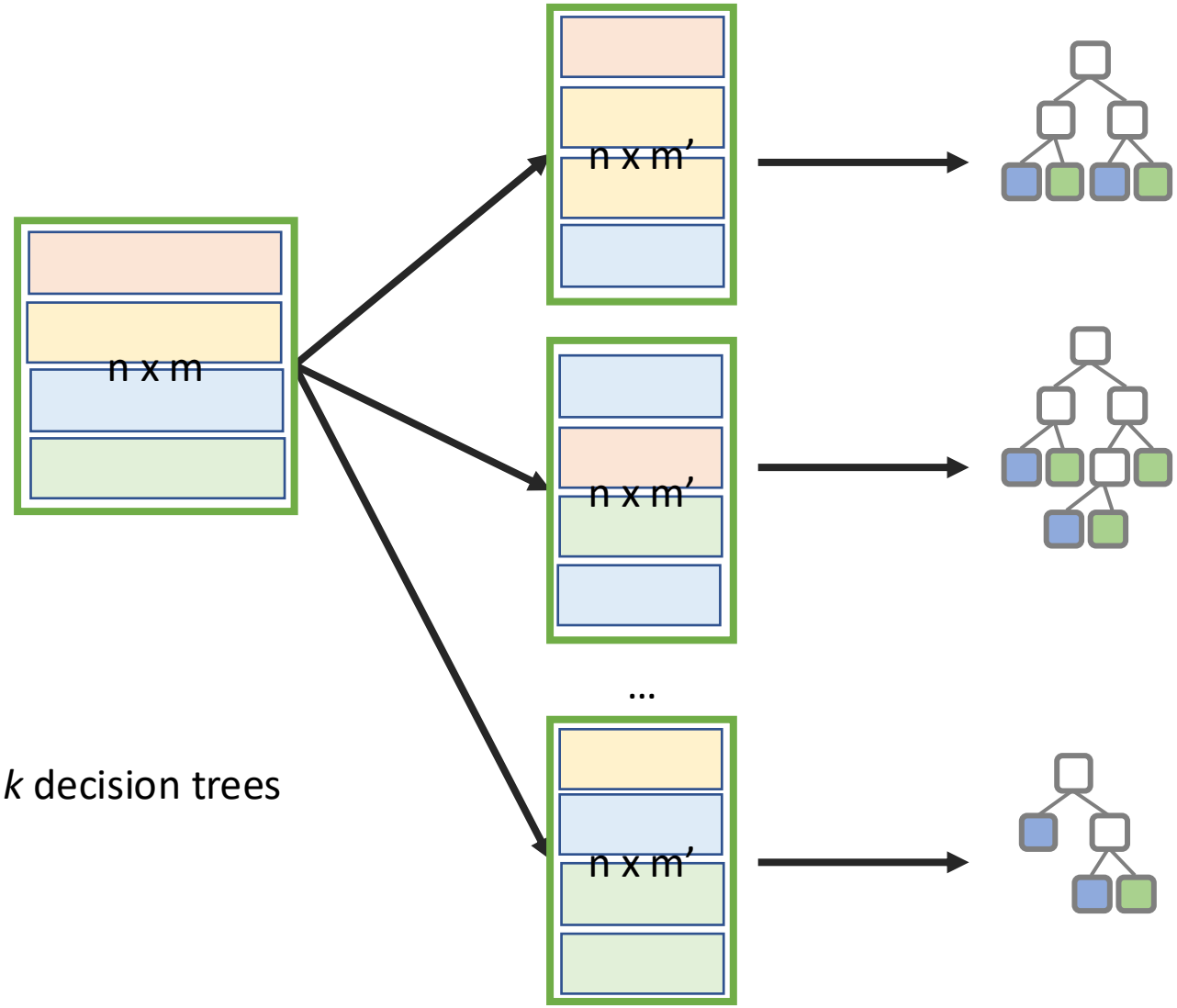
- Is a class of ensemble methods specifically designed for **decision trees**.
- It combines the predictions made by multiple decision trees and outputs the class that is the mode of the class's output by individual trees.



Random Forest

- Each decision tree is built on a **bootstrap sample** based on the values of an **independent** set of random vectors.
  - Unlike AdaBost (see next slides), the random vector are generated from a fixed probability distribution.
  - Bagging using decision trees is a special case of random forests where randomness is injected into the model-building process.
- Each decision tree is built on ***m'* randomly chosen attributes** from the *m* available attributes
  - $m' \sim \sqrt{m}$, or
  - $m' \sim \log(m)$

Random Forest



k decision trees

Random Forest - Advantages

- It is one of the most accurate learning algorithms available. For many data sets, it produces a high accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.

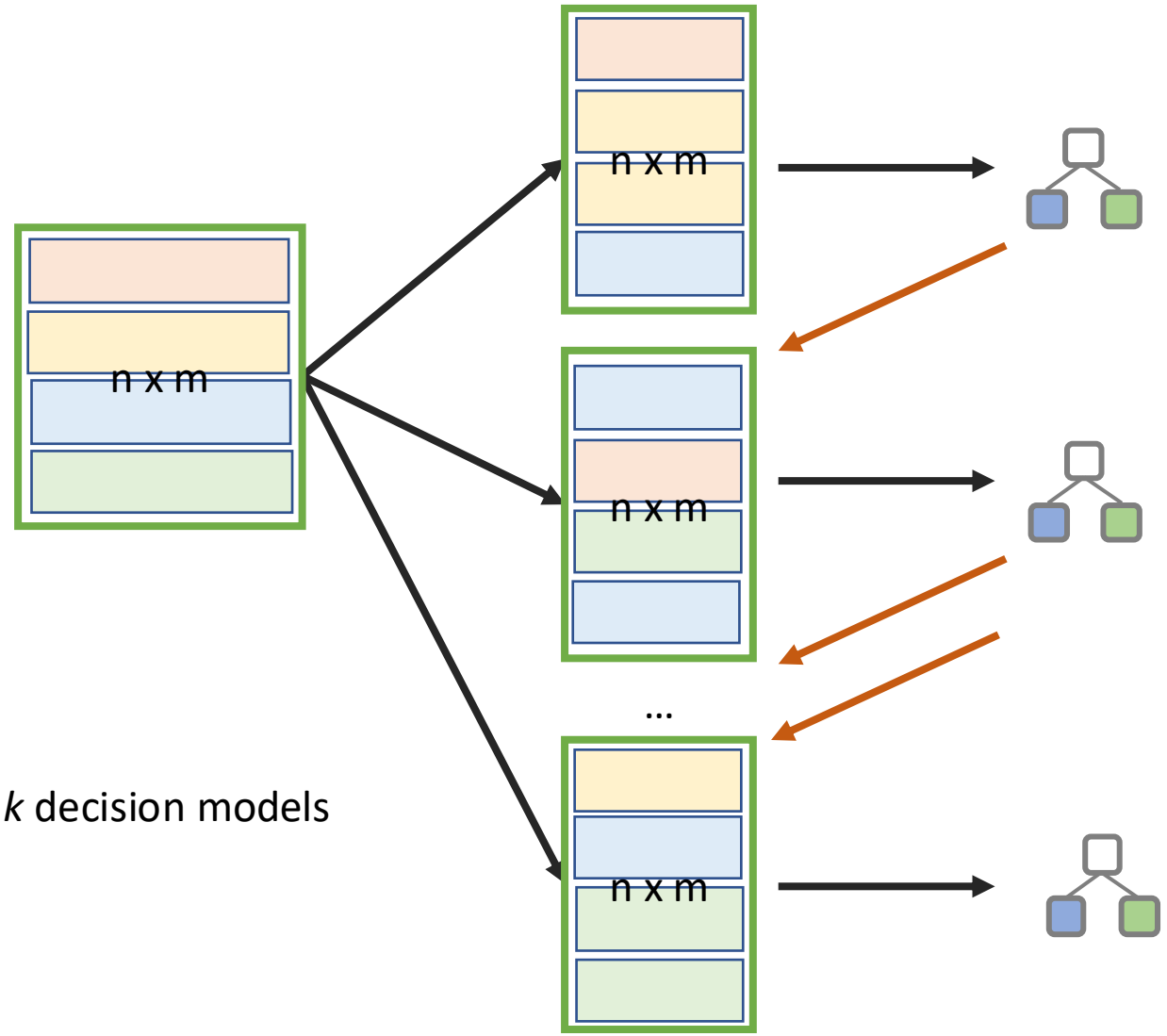
Boosting

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.

- Initially, all the records are assigned equal weights.
- Unlike bagging, weights may change at the end of each boosting round.

Boosting



k decision models

Boosting

- Records that are wrongly classified will have their weights increased.
- Records that are classified correctly will have their weights decreased.

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

AdaBoost

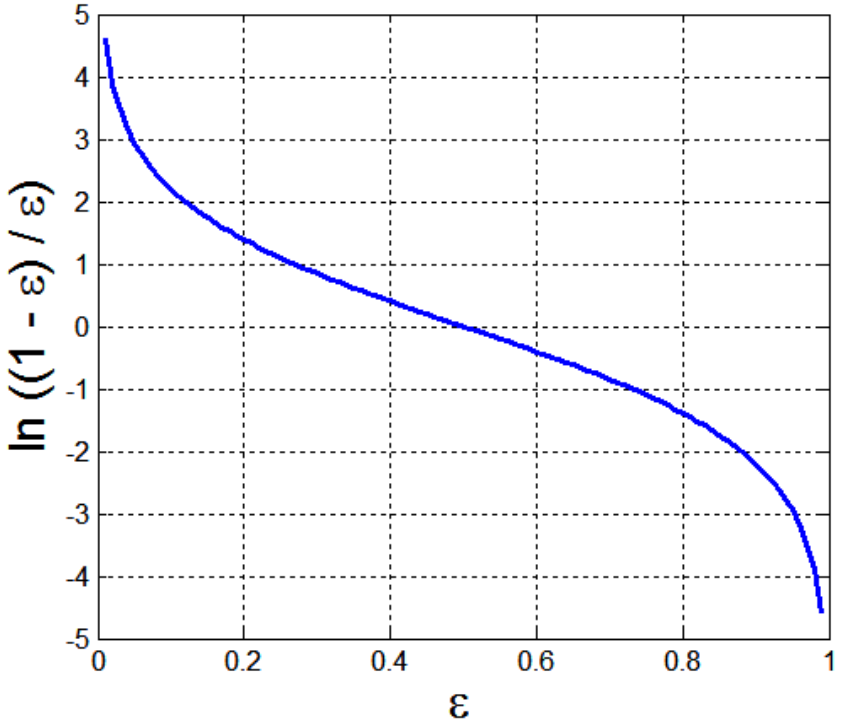
- Base classifiers: C_1, C_2, …, C_T
- Error rate:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

High positive importance when error is close to 0,
High negative importance when error is close to 1



AdaBoost Algorithm

- Weight update:

Weight associated to x_i during the j boosting round

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

- If any intermediate round produces error rate higher than 50%, the weights are reverted back to *1/n* and the resampling procedure is repeated

- Classification:

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

AdaBoost Algorithm

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to w.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j \, \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

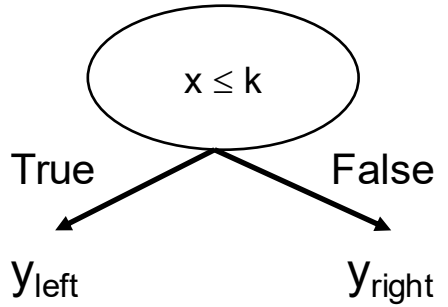
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Weights:

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

AdaBoost Example

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

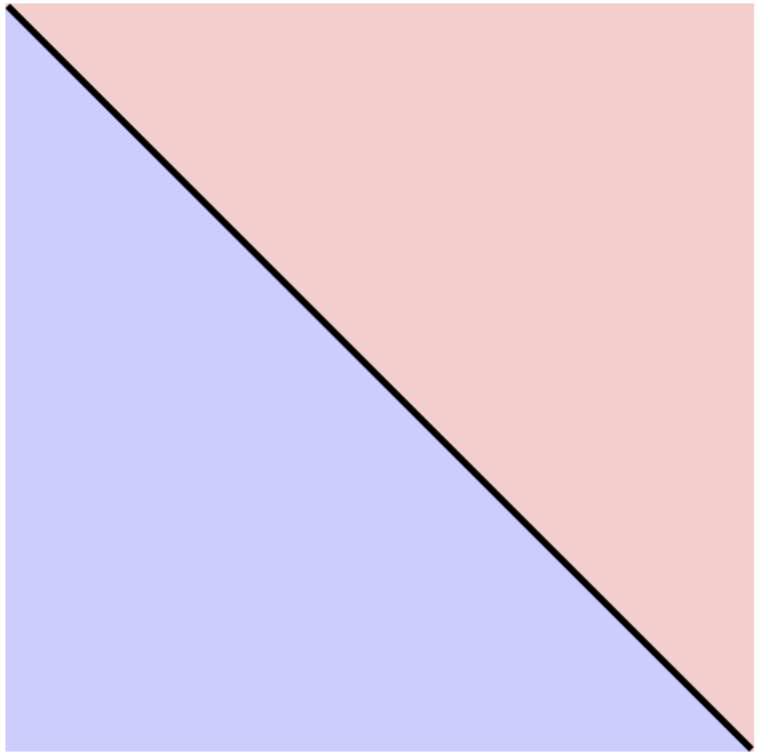
- Classification

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

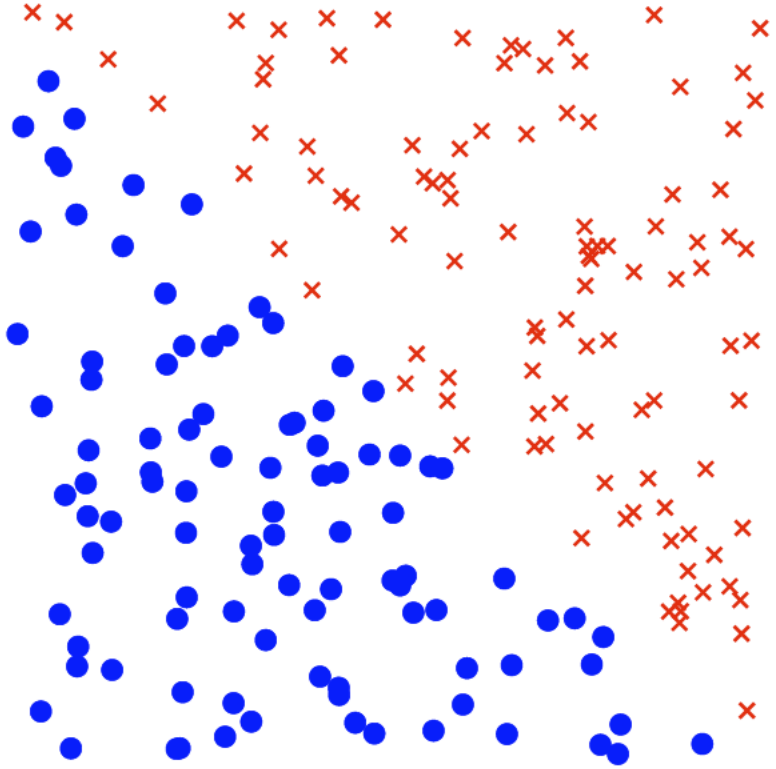
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

AdaBoost Visual Example



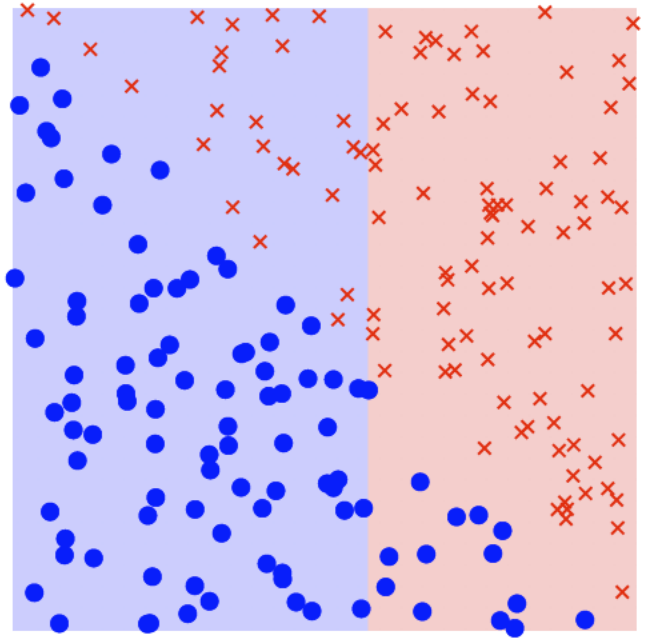
True decision boundary



Training data

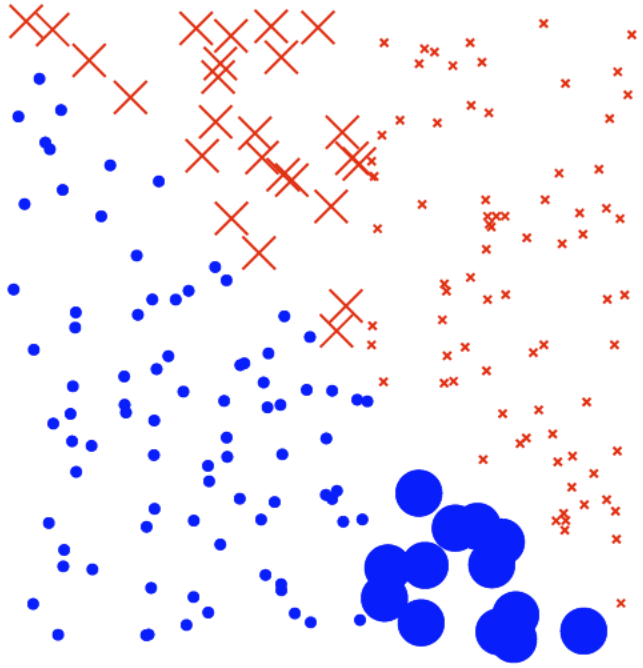
AdaBoost Visual Example

Round 1



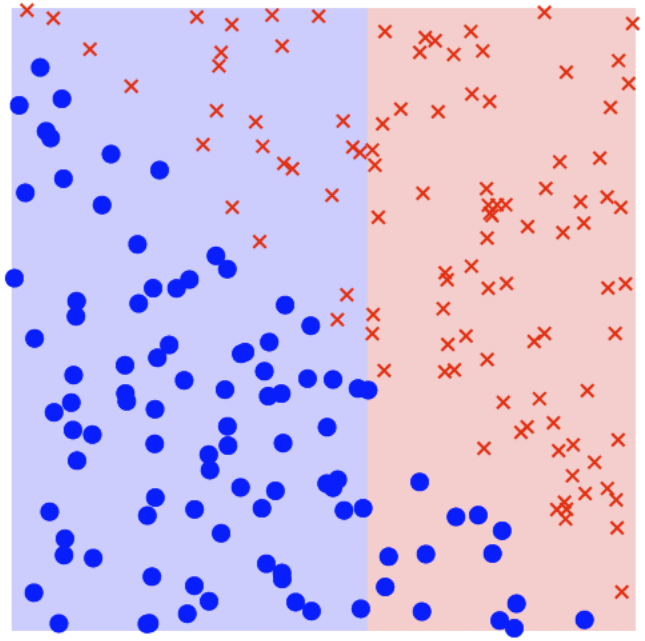
Chosen weak classifier

$\epsilon_1 = 0.19,\ \alpha_1 = 1.45$



Re-weight training points

$w_i^{(2)}$**'s**

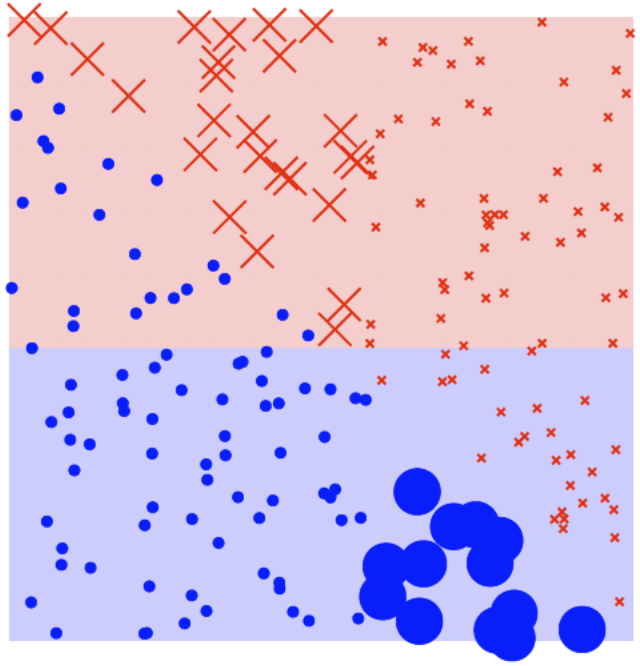


Current strong classifier

$f_2(\mathbf{x})$

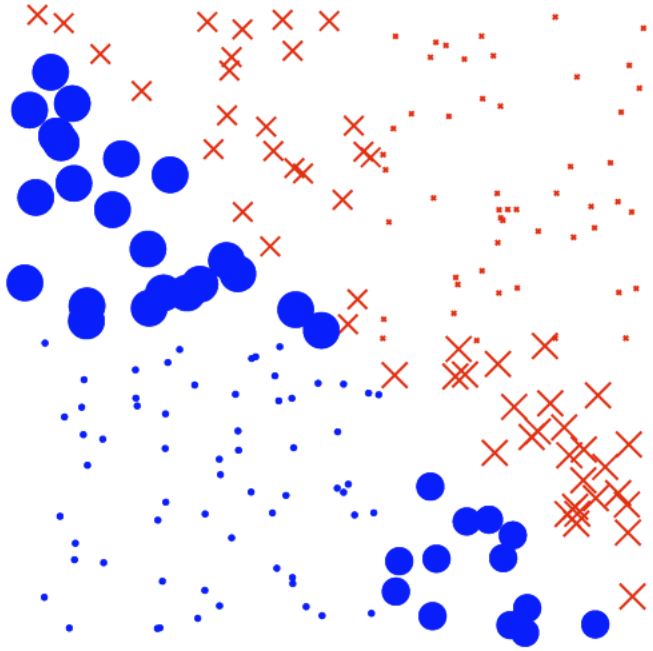
AdaBoost Visual Example

Round 2



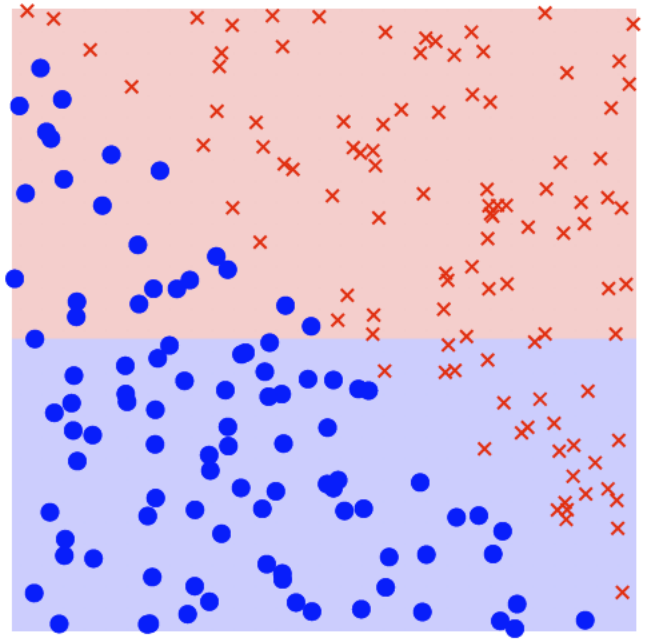
Chosen weak classifier

$\epsilon_2 = 0.1512,\ \alpha_2 = 1.725$



Re-weight training points

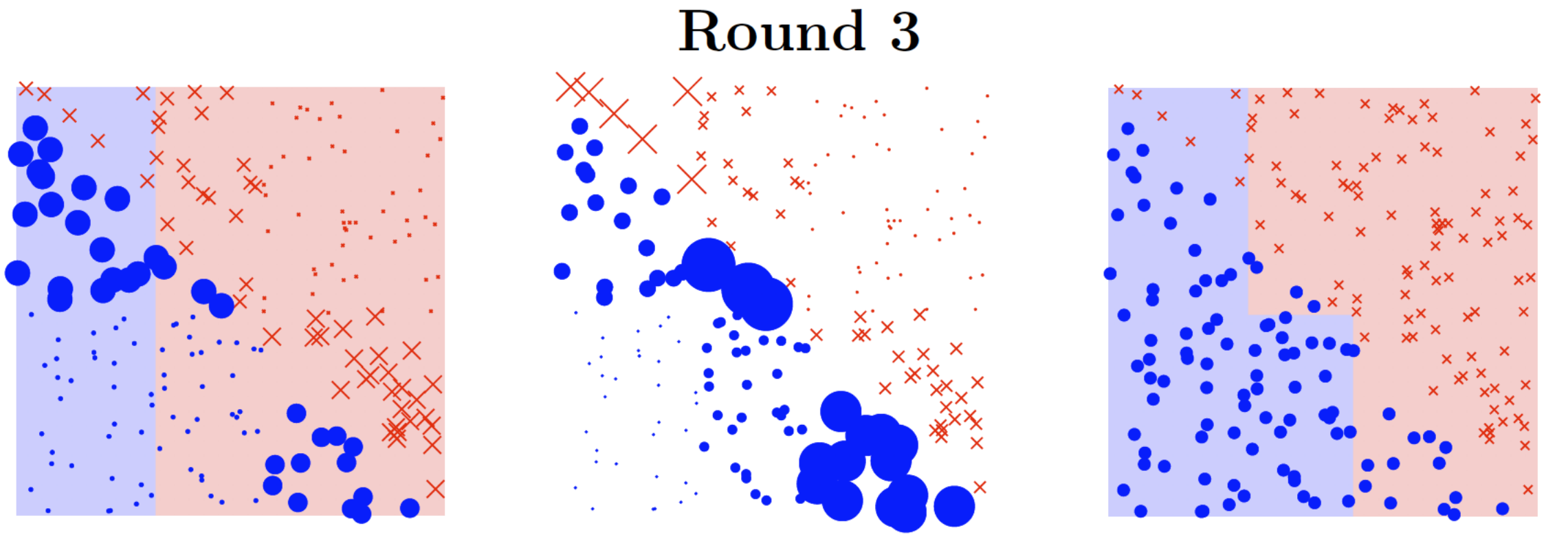
$w_i^{(3)}$**'s**



Current strong classifier

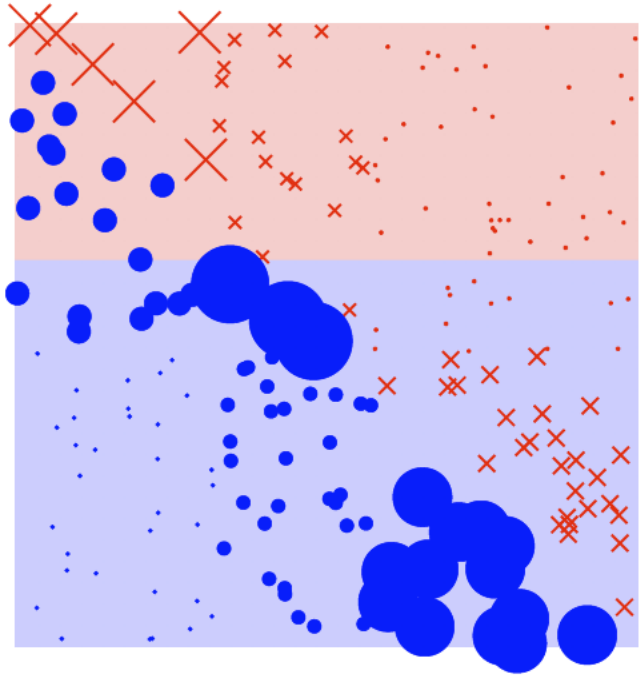
$f_2(\mathbf{x})$

AdaBoost Visual Example



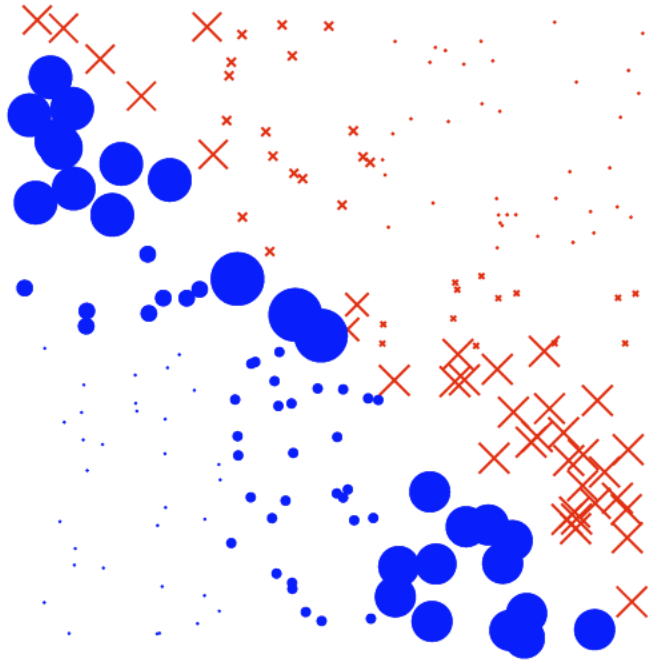
AdaBoost Visual Example

Round 4



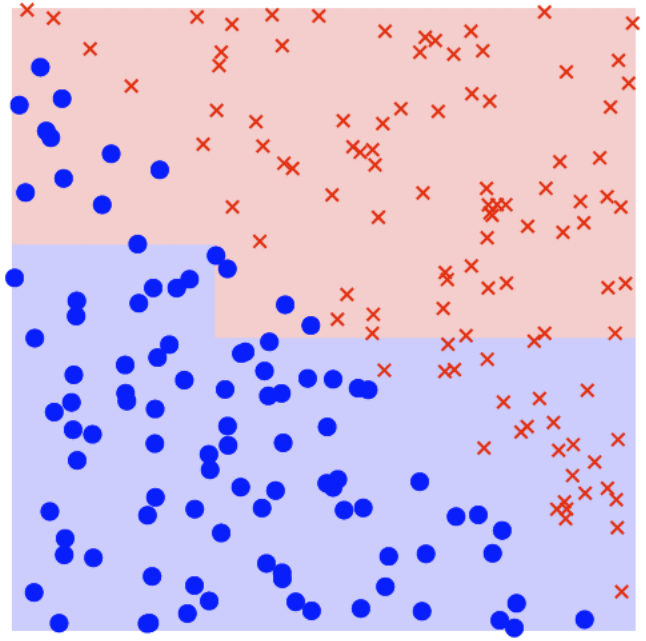
Chosen weak classifier

$\epsilon_4 = 0.2714,\ \alpha_4 = 0.9874$



Re-weight training points

$w_i^{(5)}$**'s**

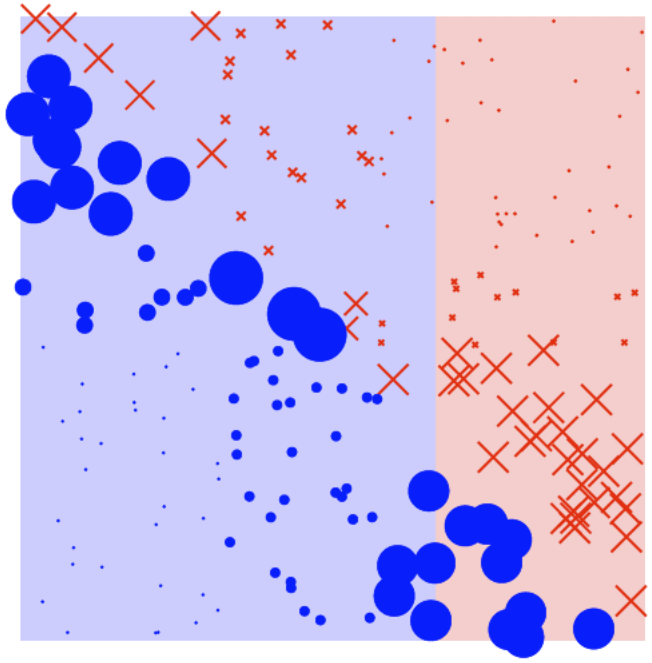


Current strong classifier

$f_4(\mathbf{x})$

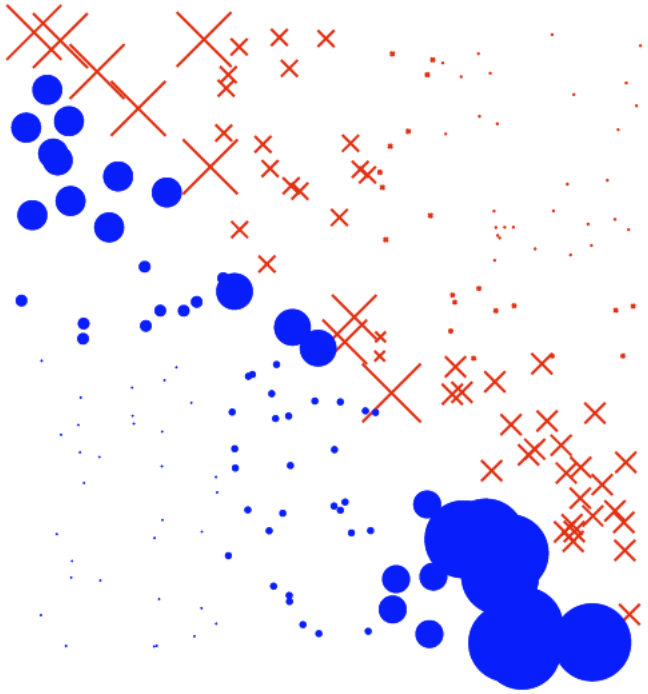
AdaBoost Visual Example

Round 5



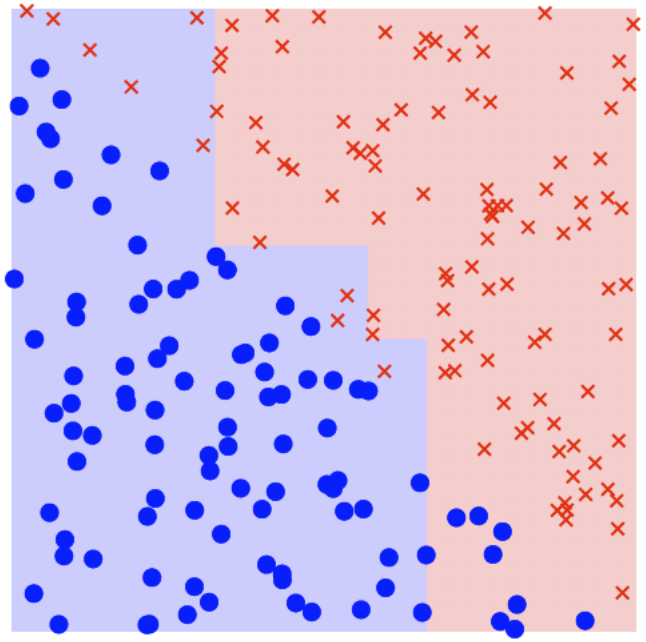
Chosen weak classifier

$\epsilon_5 = 0.2616,\ \alpha_5 = 1.0375$



Re-weight training points

$w_i^{(6)}$**'s**

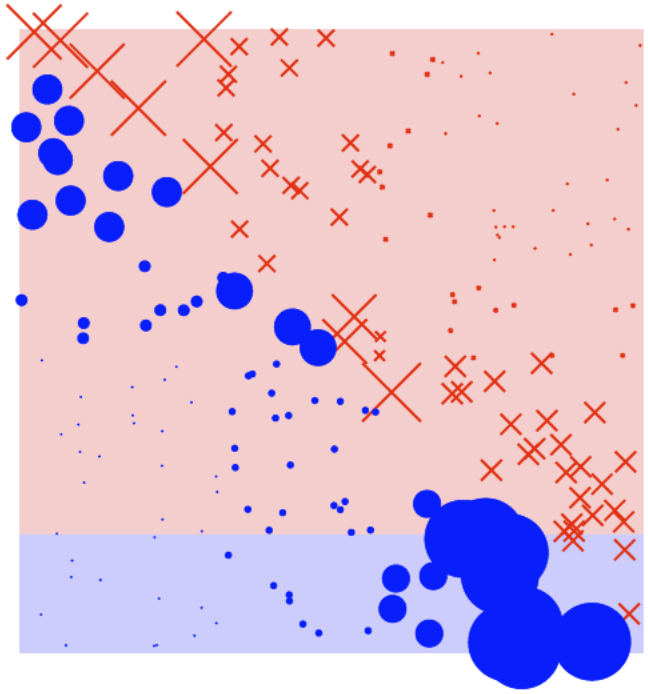


Current strong classifier

$f_5(\mathbf{x})$

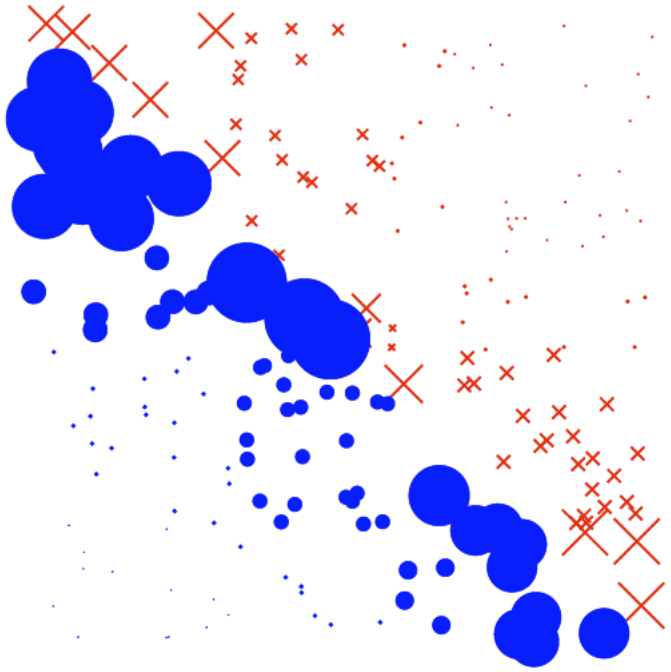
AdaBoost Visual Example

Round 6



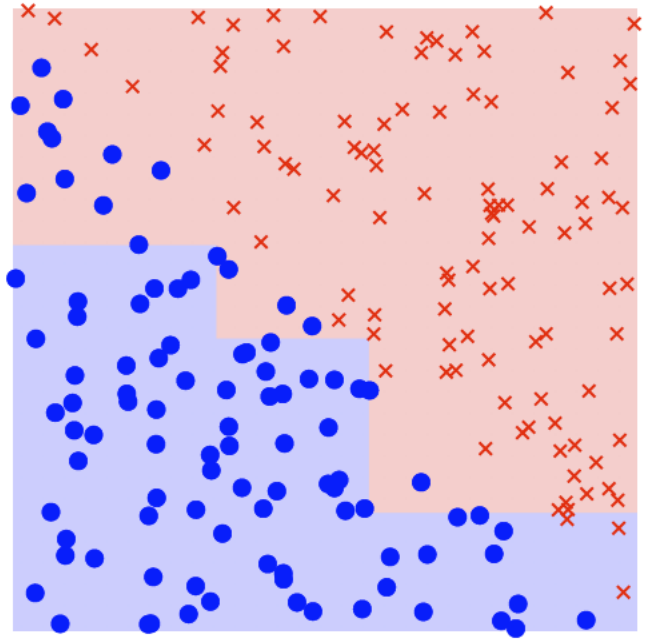
Chosen weak classifier

$\epsilon_6 = 0.2262,\ \alpha_6 = 1.2298$



Re-weight training points

$w_i^{(7)}$**'s**

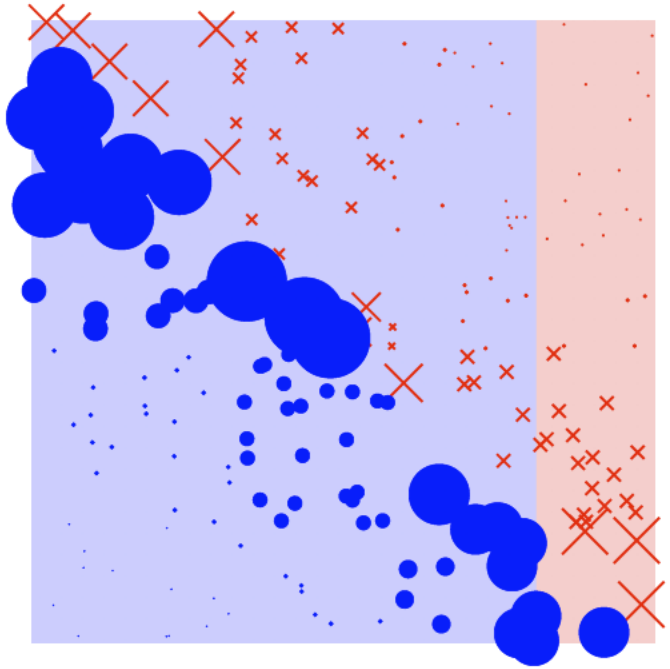


Current strong classifier

$f_6(\mathbf{x})$

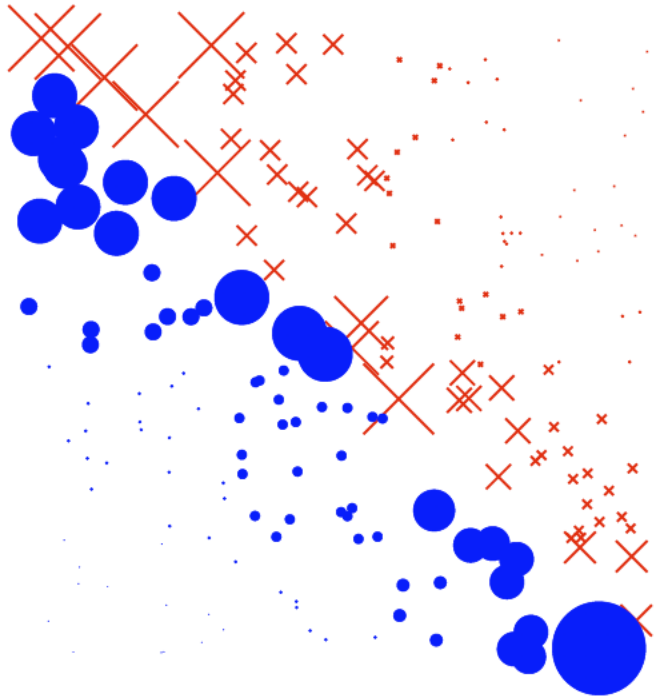
AdaBoost Visual Example

Round 7



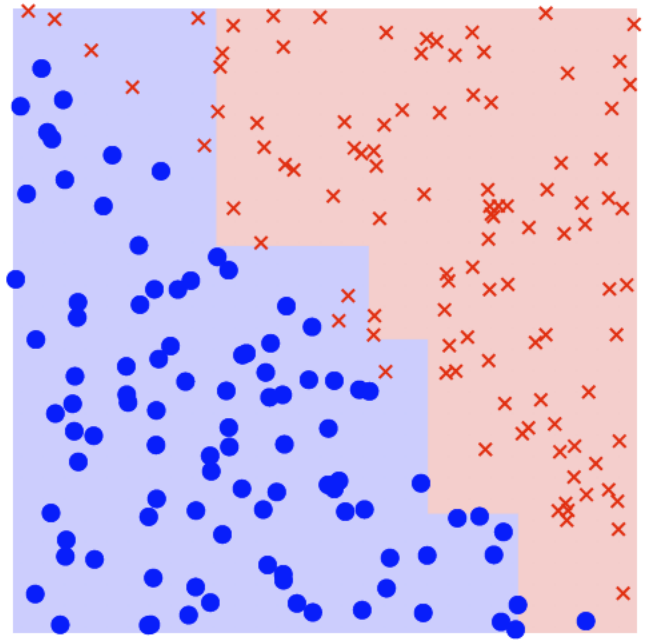
Chosen weak classifier

$\epsilon_7 = 0.2680,\ \alpha_7 = 1.0049$



Re-weight training points

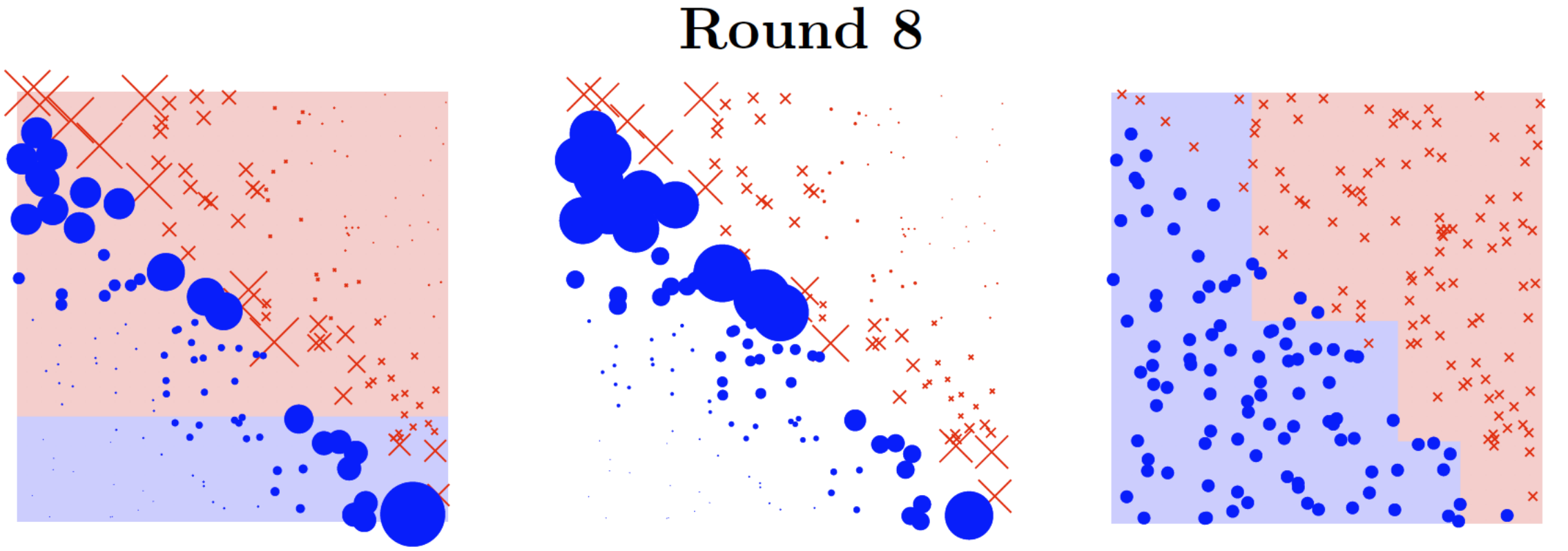
$w_i^{(8)}$**'s**



Current strong classifier

$f_7(\mathbf{x})$

AdaBoost Visual Example



Chosen weak classifier

$\epsilon_8 = 0.3282,\ \alpha_8 = 0.7165$

Re-weight training points

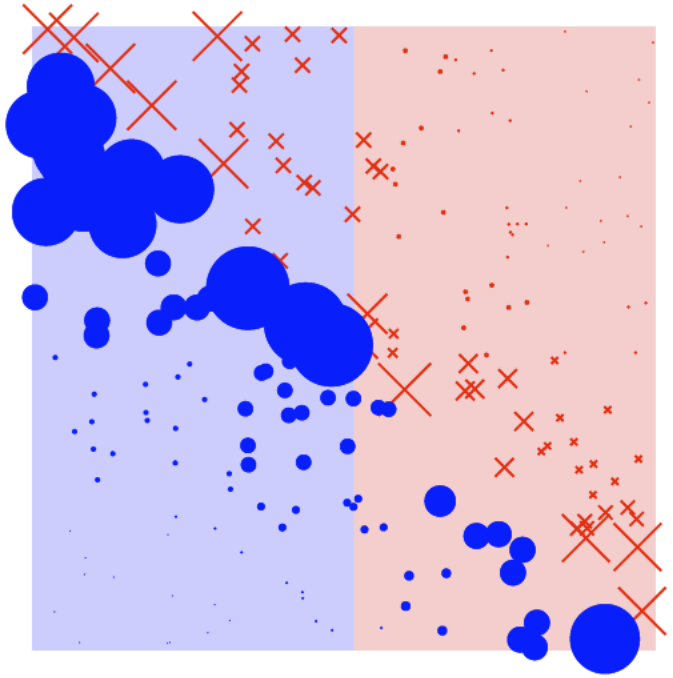
$w_i^{(9)}$**'s**

Current strong classifier

$f_8(\mathbf{x})$

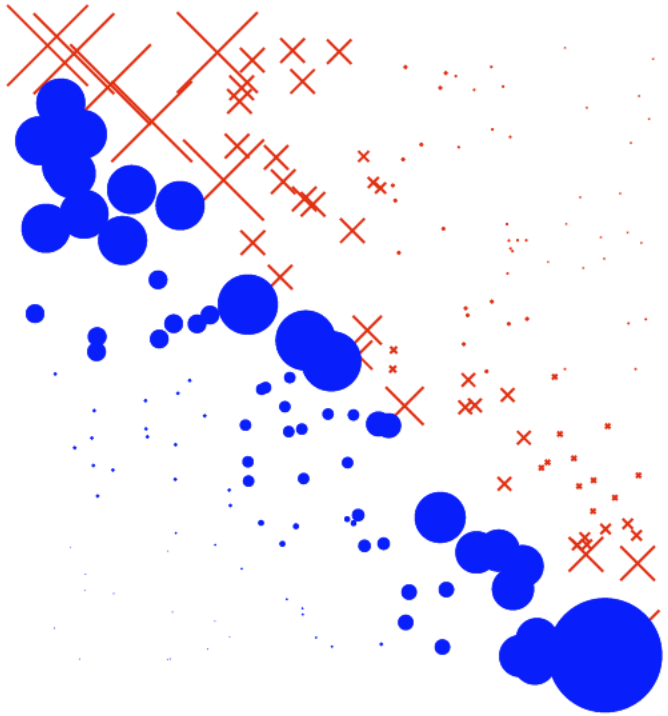
AdaBoost Visual Example

Round 9



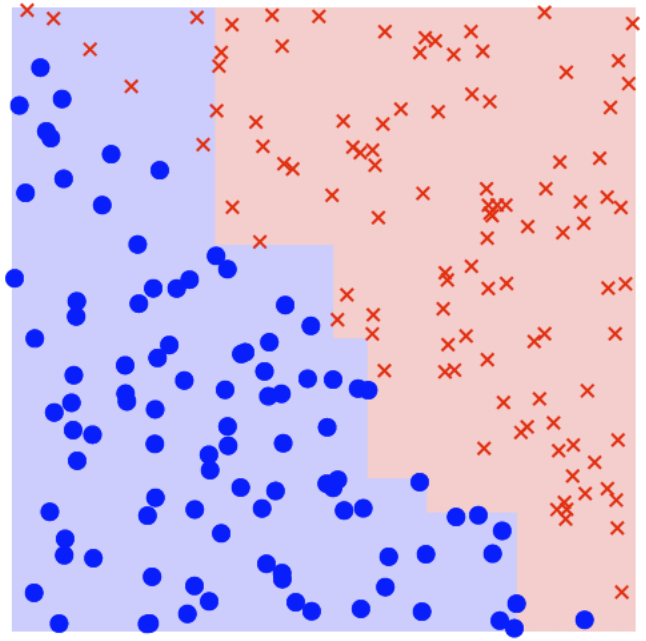
Chosen weak classifier

$\epsilon_9 = 0.3048,\ \alpha_9 = 0.8246$



Re-weight training points

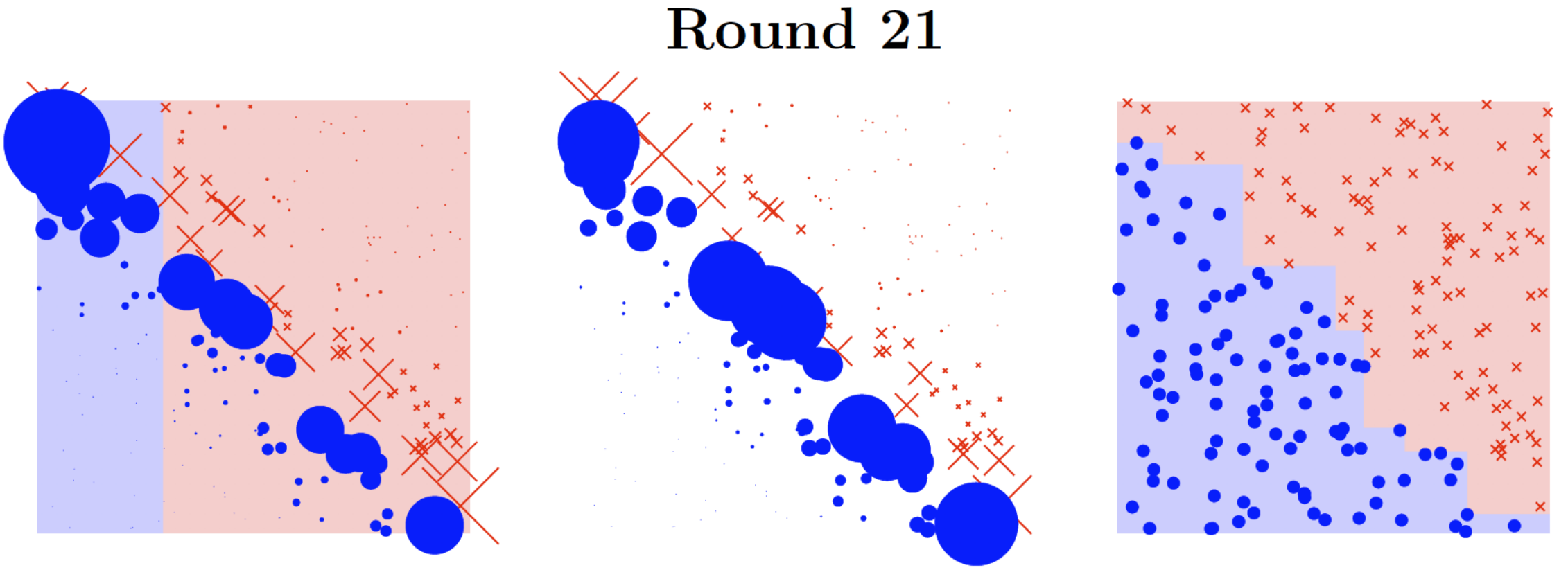
$w_i^{(10)}$**'s**



Current strong classifier

$f_9(\mathbf{x})$

AdaBoost Visual Example



Chosen weak classifier

$\epsilon_{21} = 0.3491,\ \alpha_{21} = 0.6232$

Re-weight training points

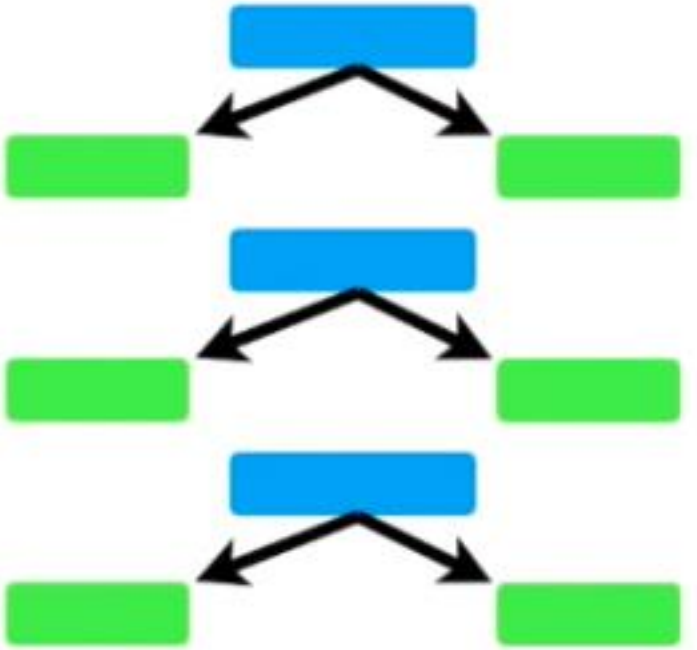
$w_i^{(22)}$**'s**

Current strong classifier

$f_{21}(\mathbf{x})$

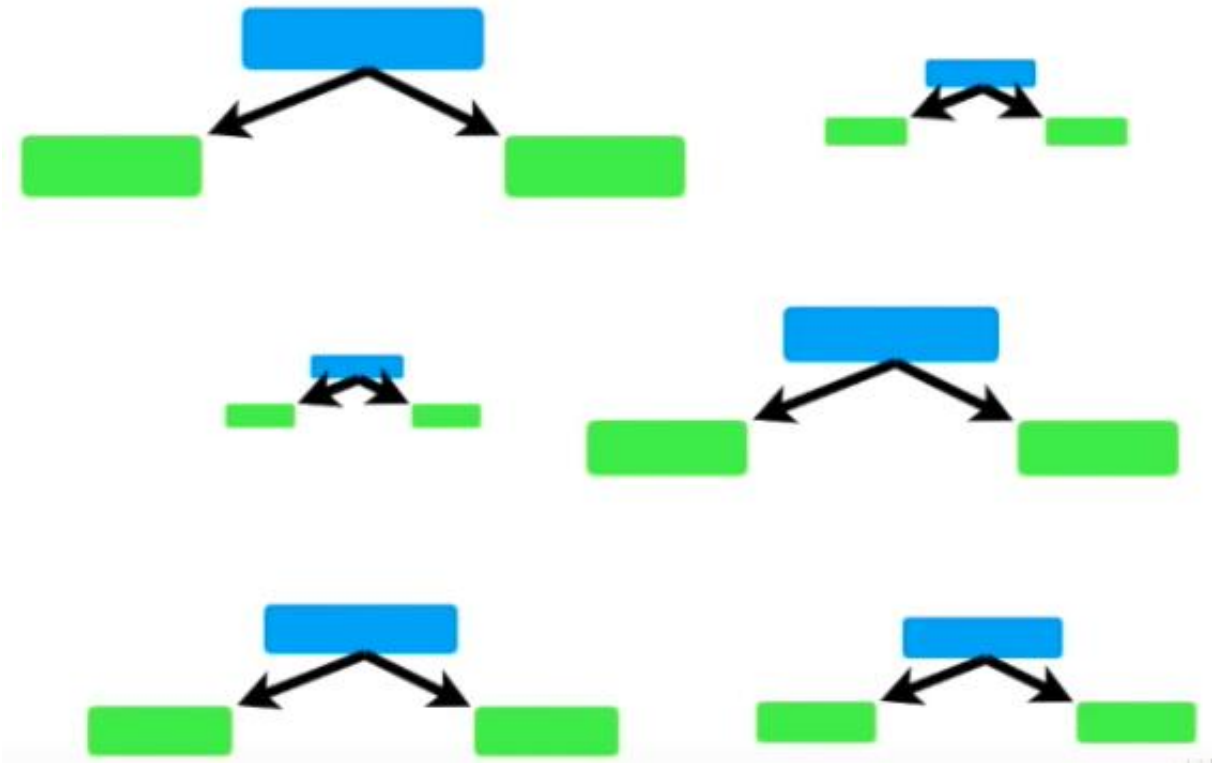
AdaBoost - Reloaded

- AdaBoost uses **stumps**, i.e., decision tree with only one node and two leaves.
- Thus, it uses a forest of stumps.
- Stumps are not good at making accurate classifications.
- Stumps are *weak* learners.



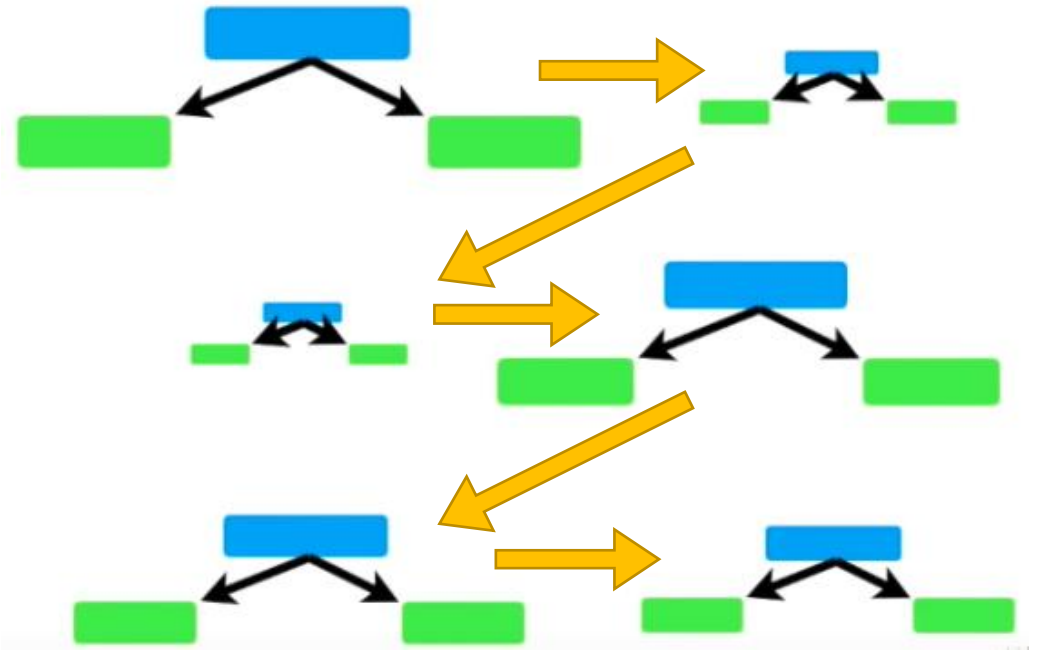
AdaBoost - Reloaded

- In forest of stumps some stumps have more importance in the final classification.



AdaBoost - Reloaded

- Each stump **is not** made independently from the others.
- The error that the first stump makes influences the second stump and so on and so forth.



AdaBoost – Training Step by Step

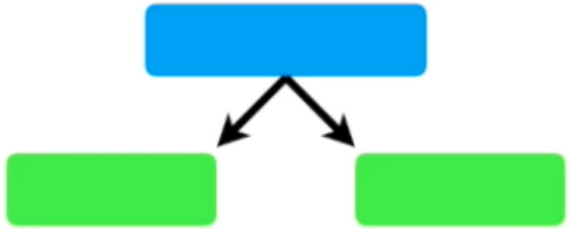
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

- Initially all the records get the same weight: 1/number of samples.

AdaBoost – Training Step by Step

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

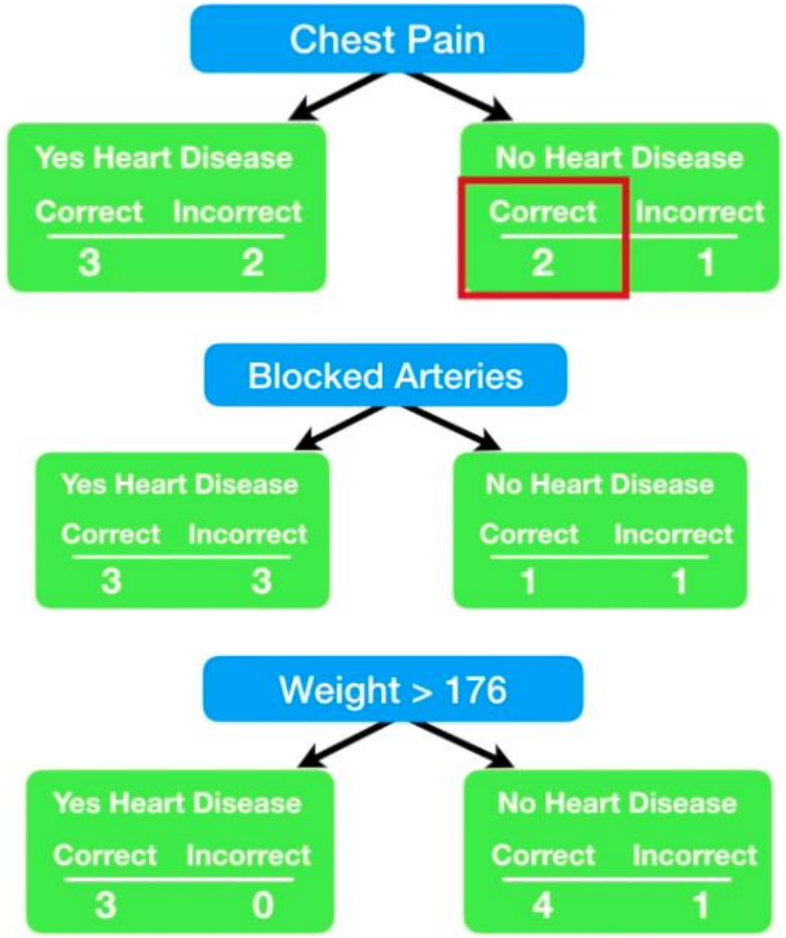
- We search for the first stump of the forest.



- All the weights are the same thus we can ignore them now.

AdaBoost – Training Step by Step

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

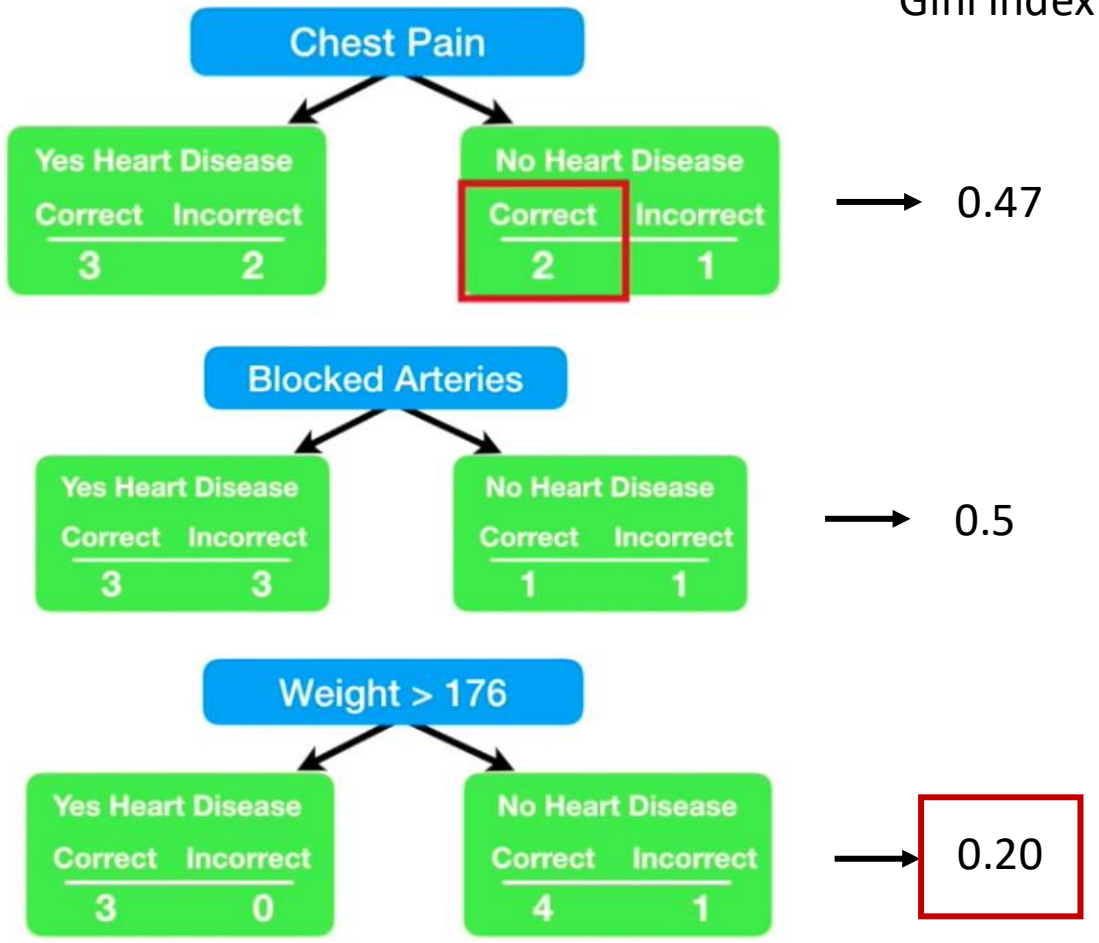


AdaBoost – Training Step by Step

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

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

Gini Index



First Stump in the Forest

AdaBoost – Training Step by Step

- Now we need to determine the importance of this stump in the final classification.
- We determine the importance based on how well it classified the samples in terms of *total error* in the following formula:
- The total error is always between 0 and 1.

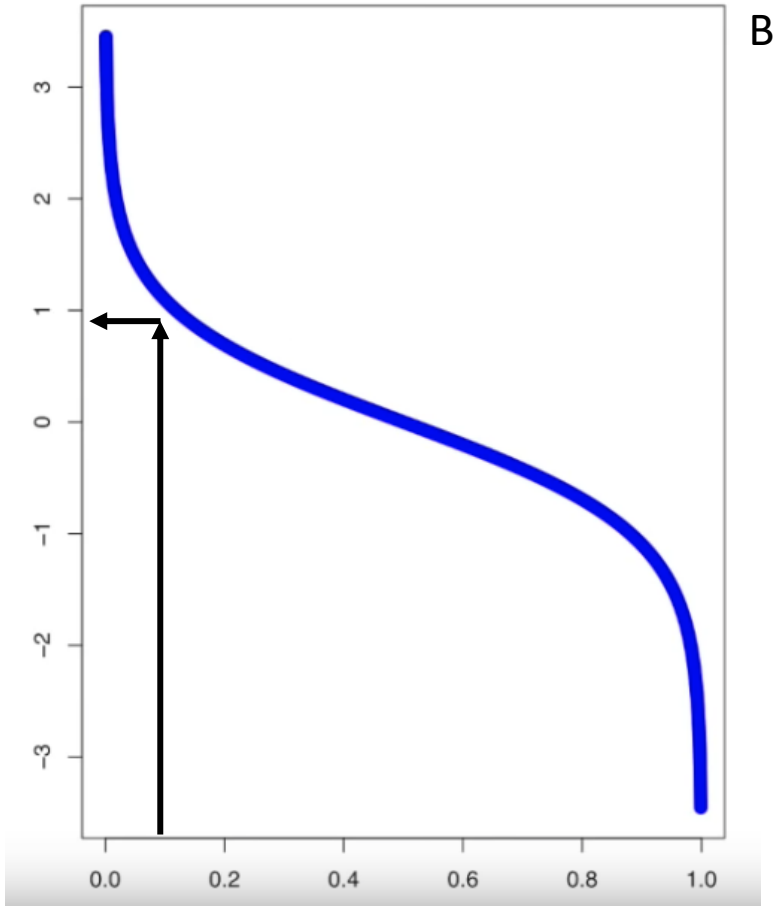


Total Error = 1/8

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

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

AdaBoost – Training Step by Step



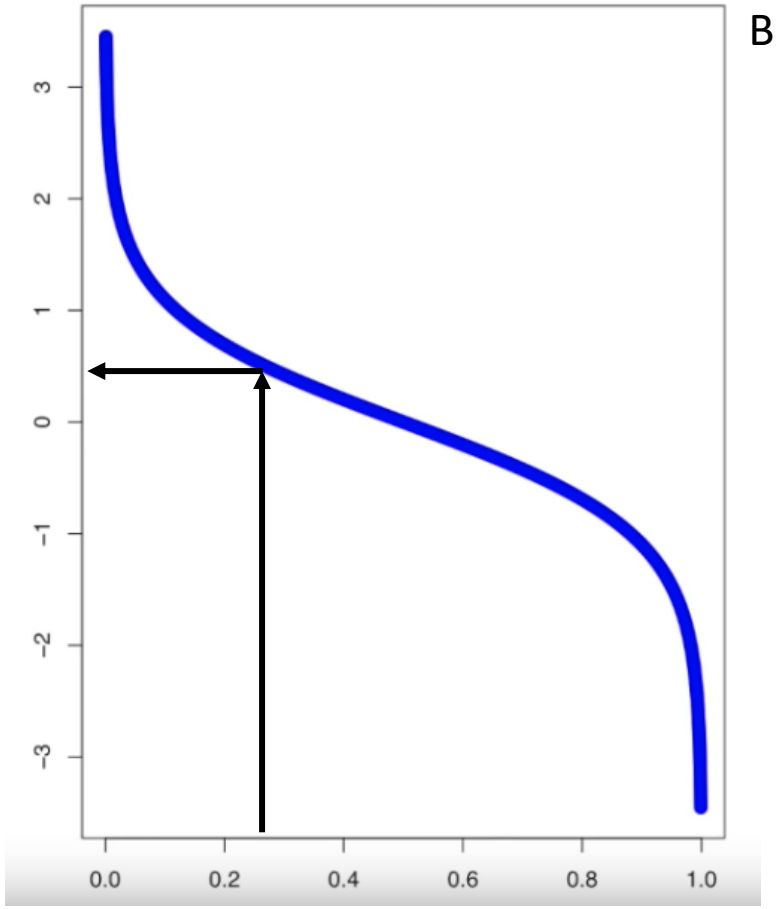
Total Error

Blue line is the Importance

- In our case the importance is 0.97 = 1/2 log(7)



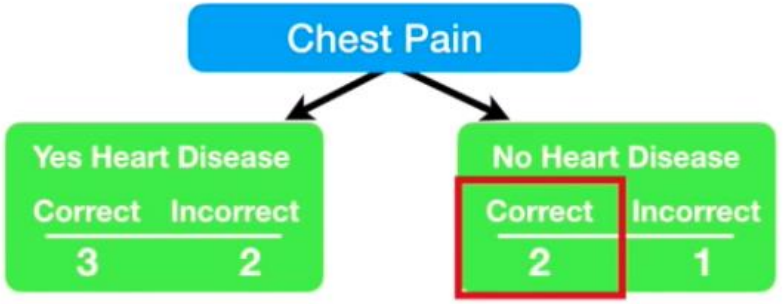
AdaBoost – Training Step by Step



Total Error

Blue line is the Importance

- If it had been Chest Pain
- Total error = 3/8
- Importance 1/2 log(7/3) = 0.42



AdaBoost – Training Step by Step

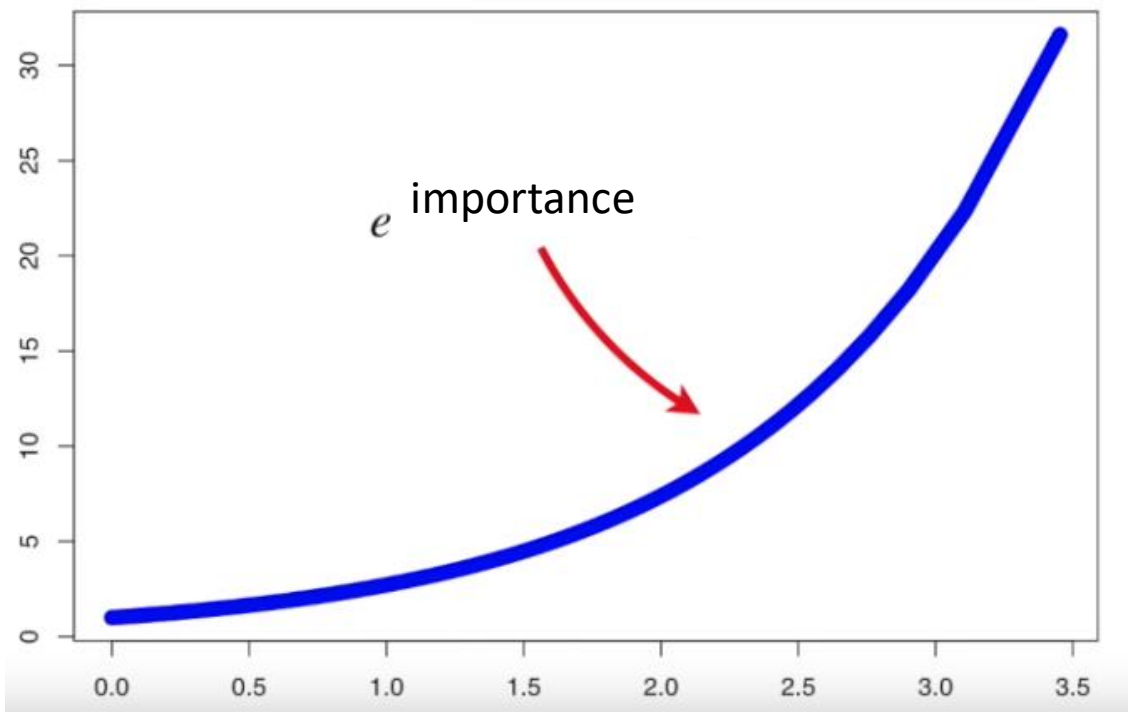
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

- Update weights.
- We need to emphasize the usage of this sample in the next iteration by increasing its weight and decreasing the others.

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{importance}}$$



AdaBoost – Training Step by Step



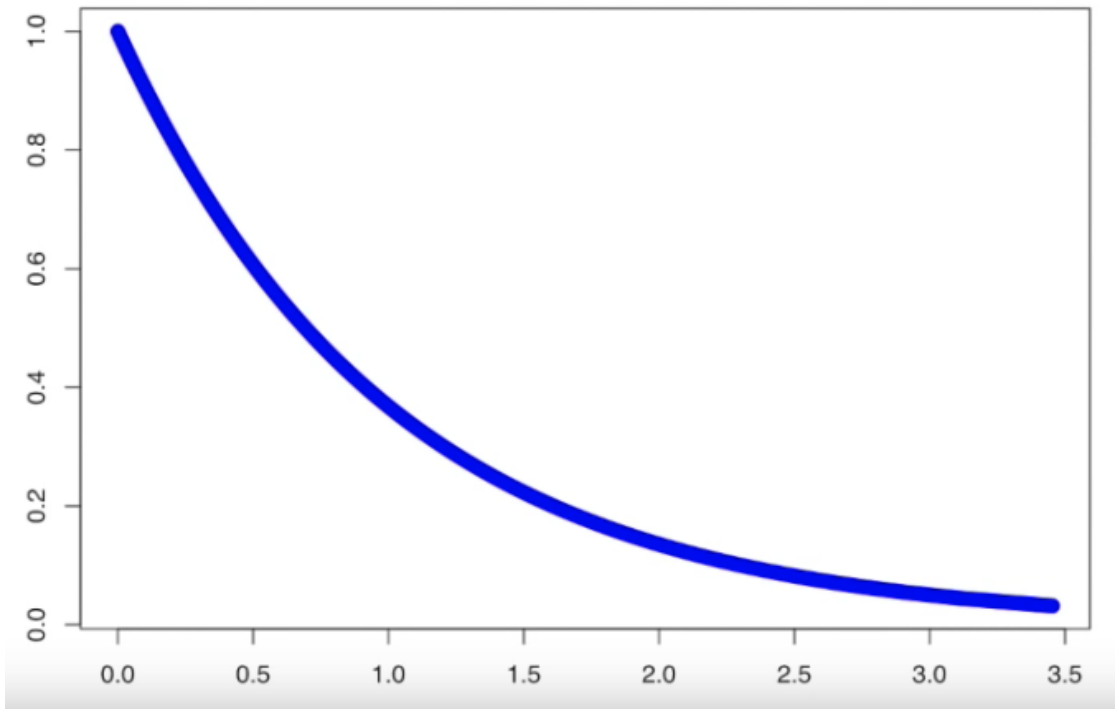
$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{importance}}$$

$= 1/8\ e^{0.97} = 1/8 * 2.64 = 0.33 > 0.125 = 1/8$

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

AdaBoost – Training Step by Step



importance

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{-importance}}$$

$= 1/8\ e^{-0.97} = 1/8 * 0.38 = 0.05 < 0.125 = 1/8$

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

AdaBoost – Training Step by Step

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

- Report the new weights.
- Normalize the new weights so that they sum up to 1.
- Replace the old ones.

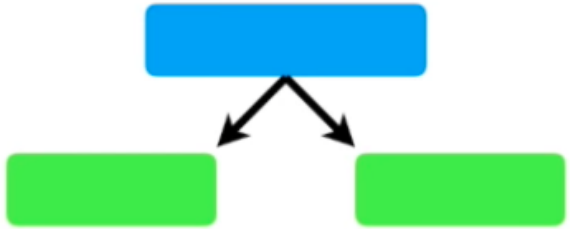
$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

AdaBoost – Training Step by Step

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

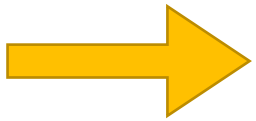
- Now we can use the modified sample weight to make the second stump in the forest.



- In theory we could use weighted Gini Index to determine which variable should split the next stump.
- Alternatively, we can make a new training set that contains duplicate copies of the samples with the largest sample weights.

AdaBoost – Training Step by Step

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



Then we use the new collection of samples to repeat the process.

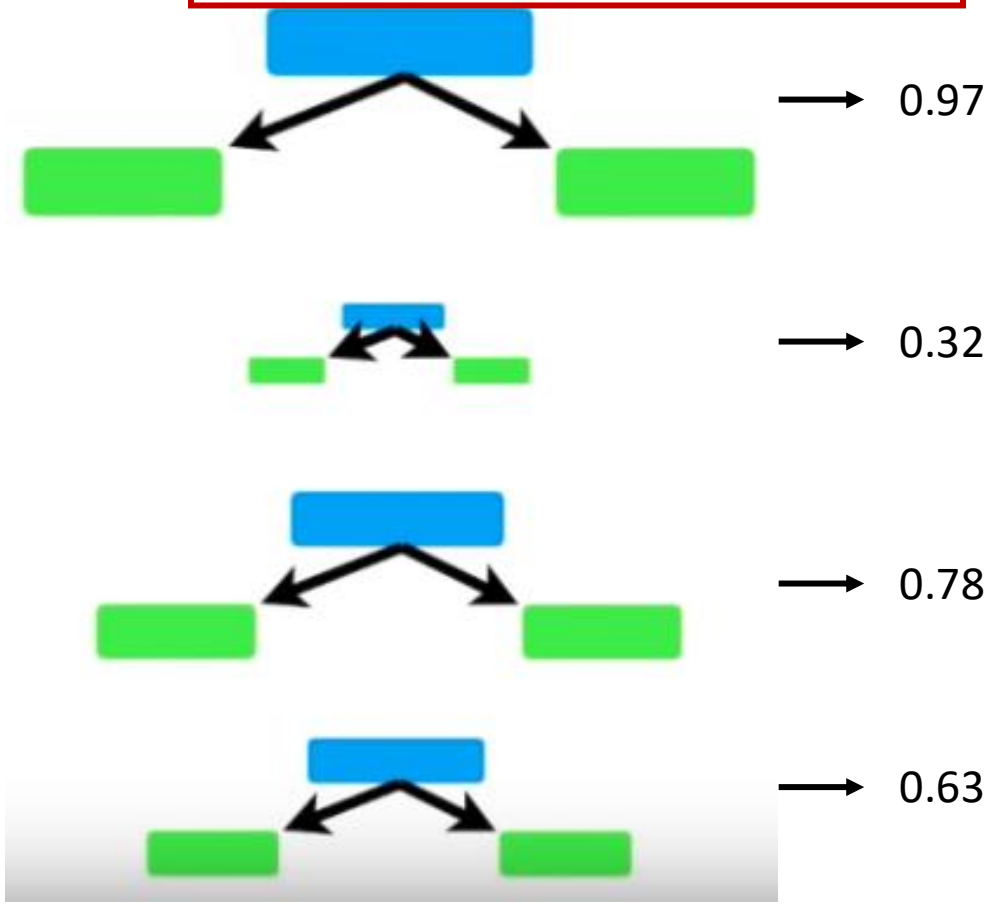
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

AdaBoost – Classification

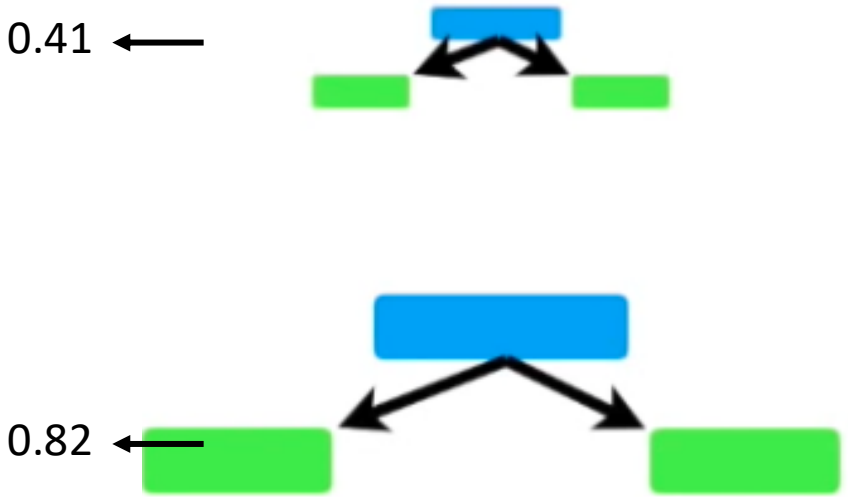
Given a patient

Chest Pain	Blocked Arteries	Patient Weight
Yes	Yes	205

Heart Disease = Yes Total = 2.7



Total = 1.23 Heart Disease = No



Gradient Boosting for Regression

Gradient Boost – Main Idea

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

Gradient Boost – Main Idea

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



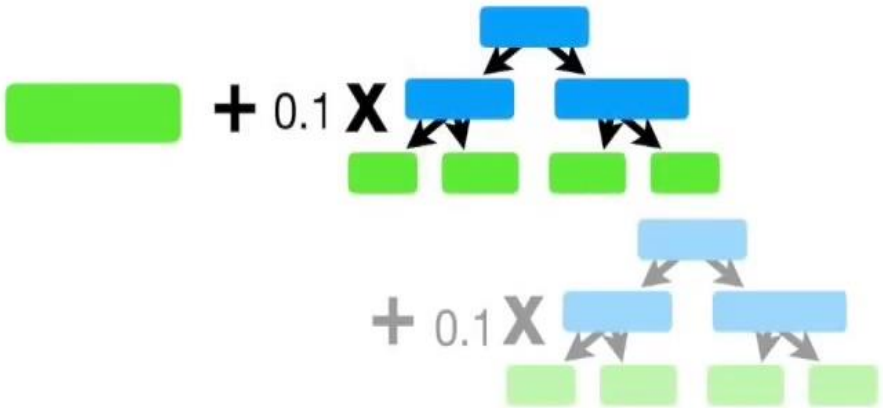
Gradient Boost – Main Idea

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



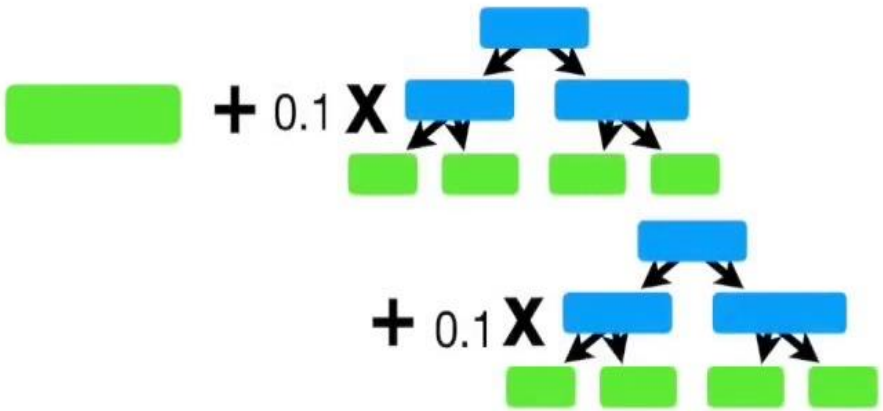
Gradient Boost – Main Idea

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



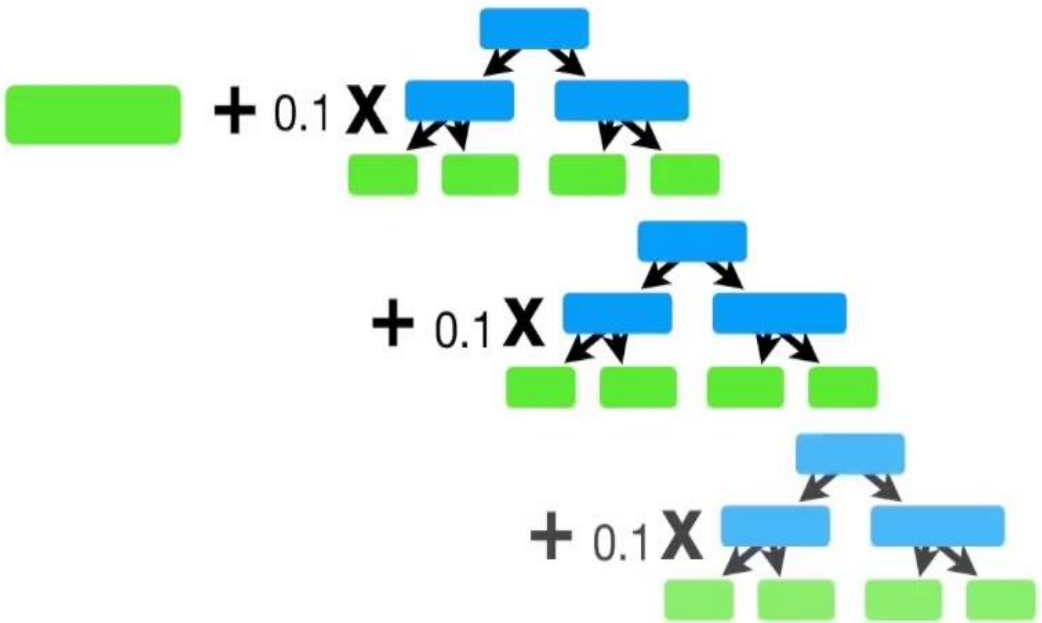
Gradient Boost – Main Idea

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



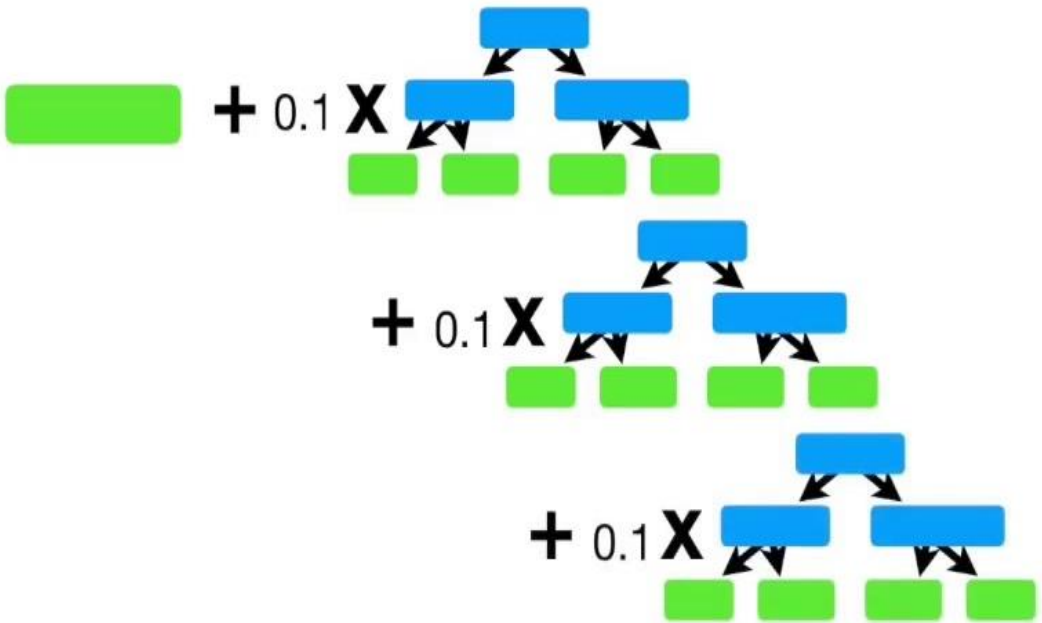
Gradient Boost – Main Idea

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

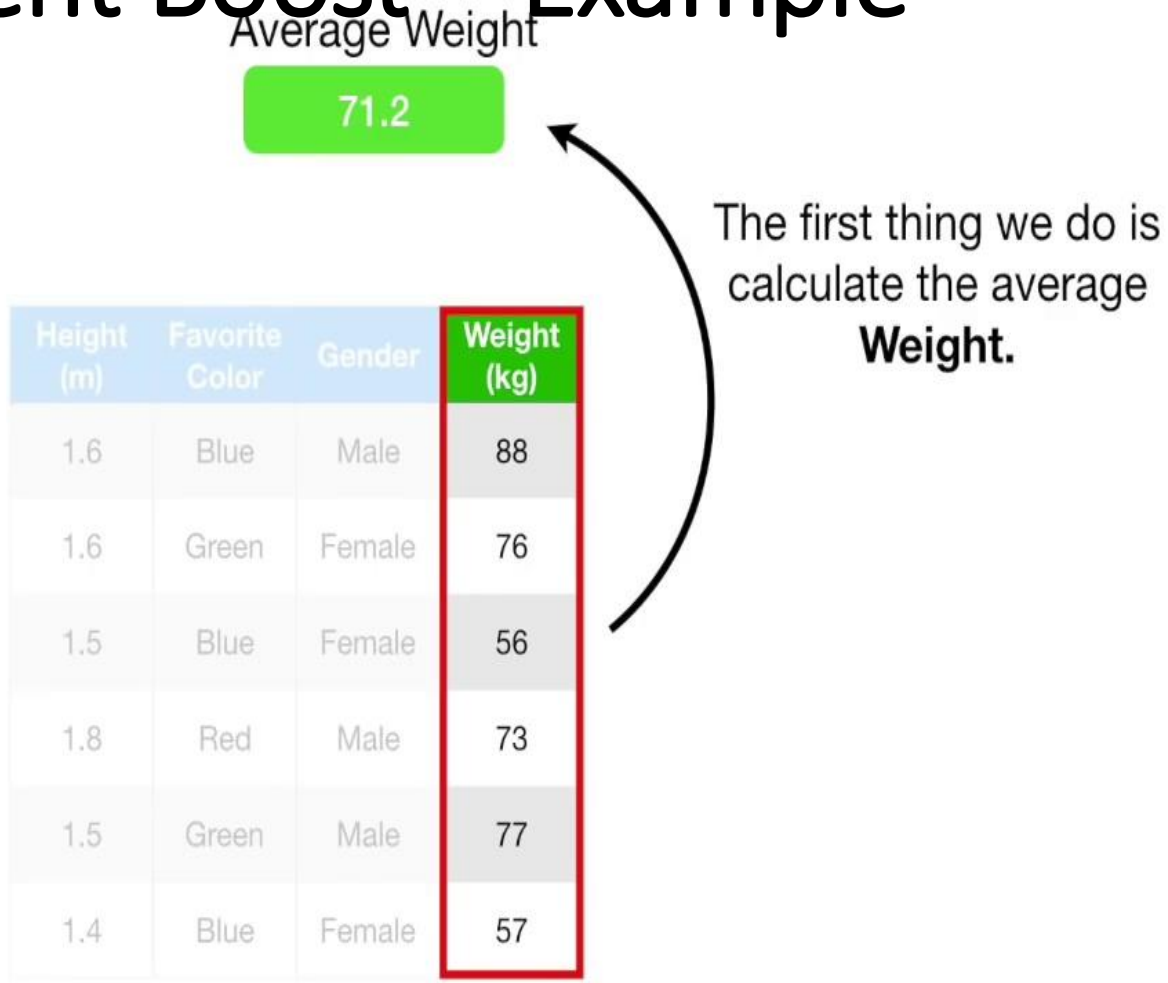


Gradient Boost – Main Idea

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



Gradient Boost – Example



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

Example

Average Weight

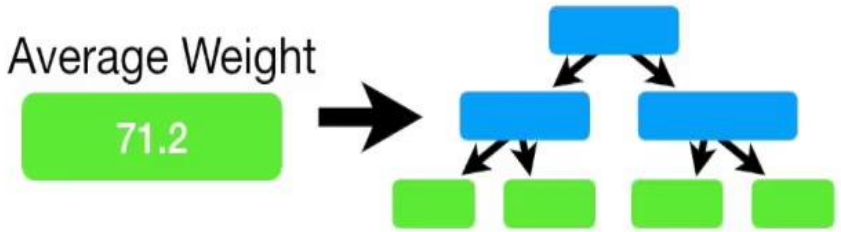
71.2



This is the first attempt at predicting everyone's 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

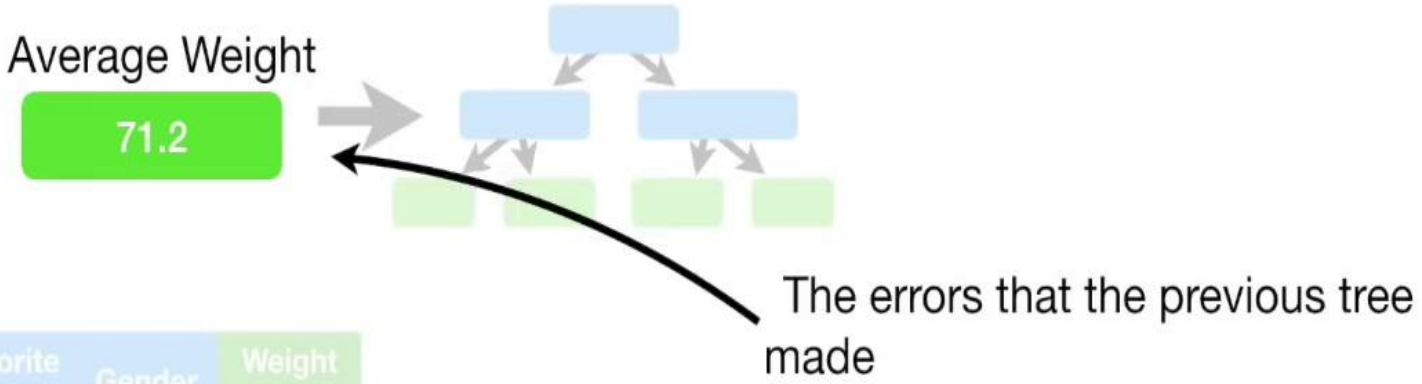
Example



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

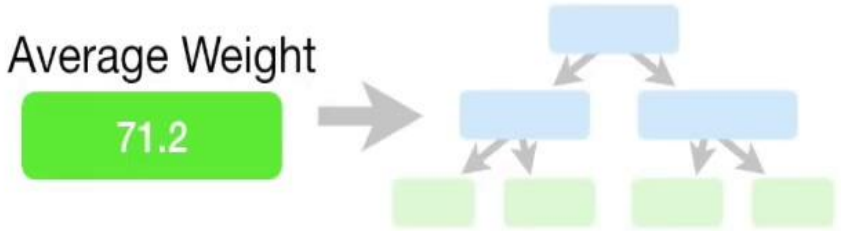
The next thing we do is build a tree based on the errors from the first tree.

Example



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

Example

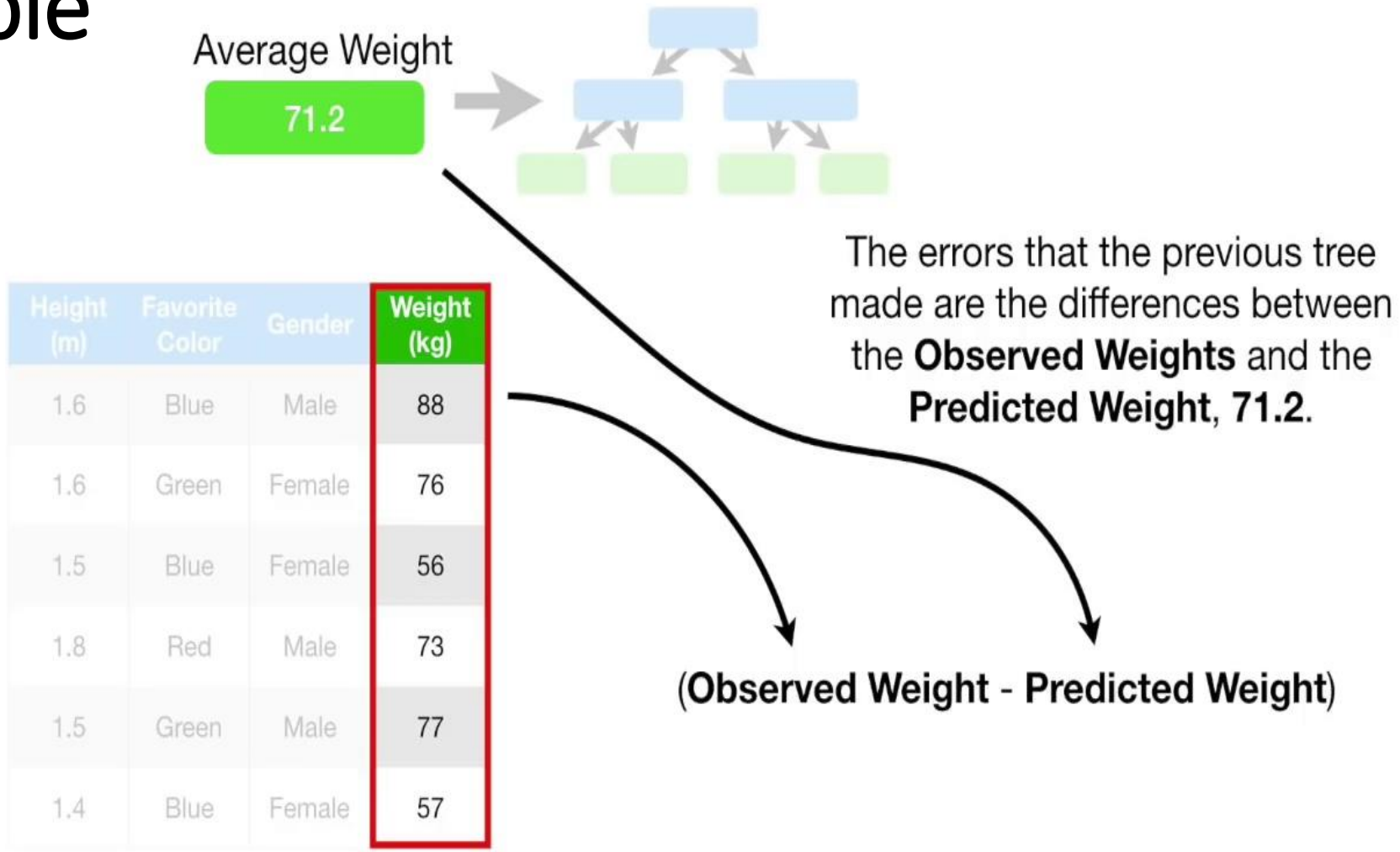


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

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

(**Observed Weight** - Predicted Weight)

Example



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

Example

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	

…and save the difference, which is called a **Pseudo Residual**, in a new column.

(**88** - **71.2**) = 16.8

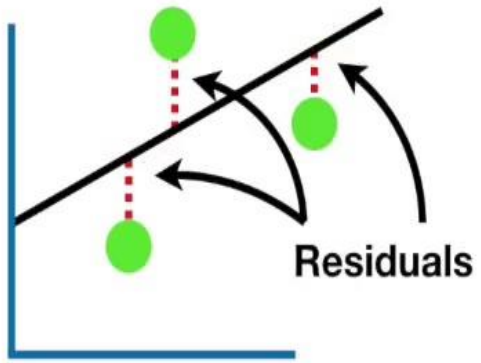
Example

Average Weight

71.2

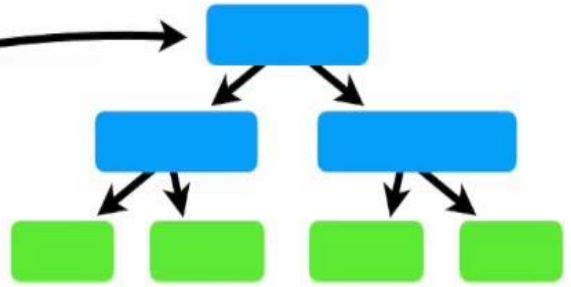
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	
1.4	Blue	Female	57	

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



Example

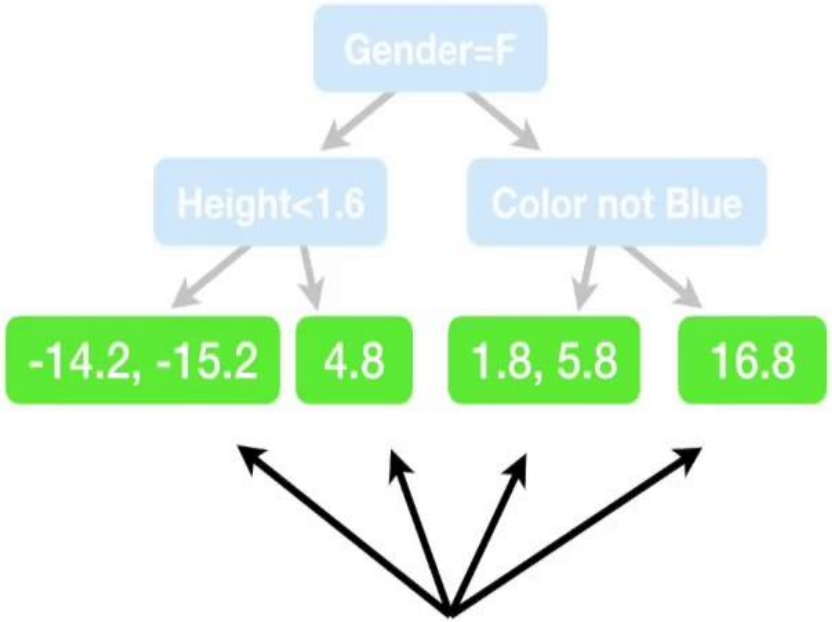
Now we will build a **Tree**



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

Example

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

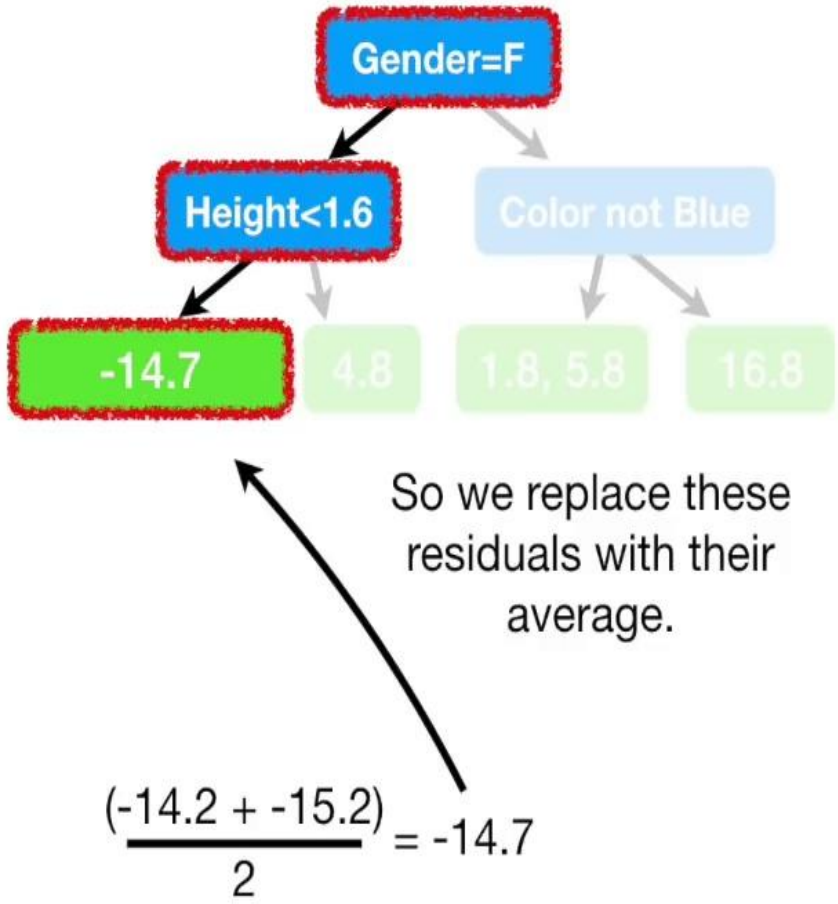


Remember, in this example we are only allowing up to four leaves…

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

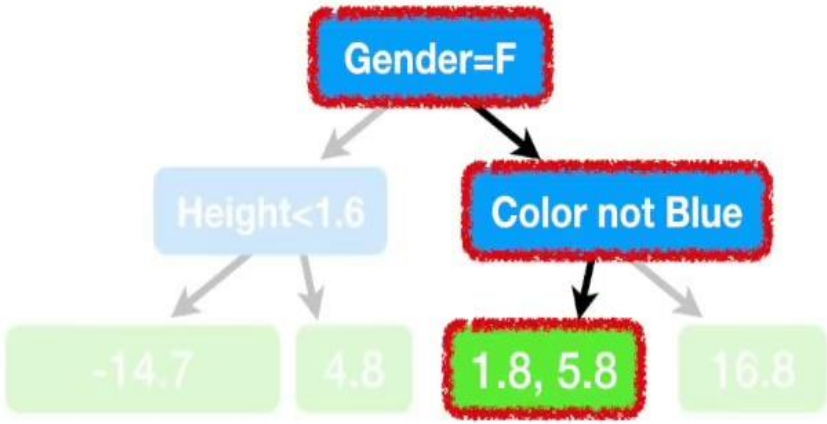
Example

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



Example

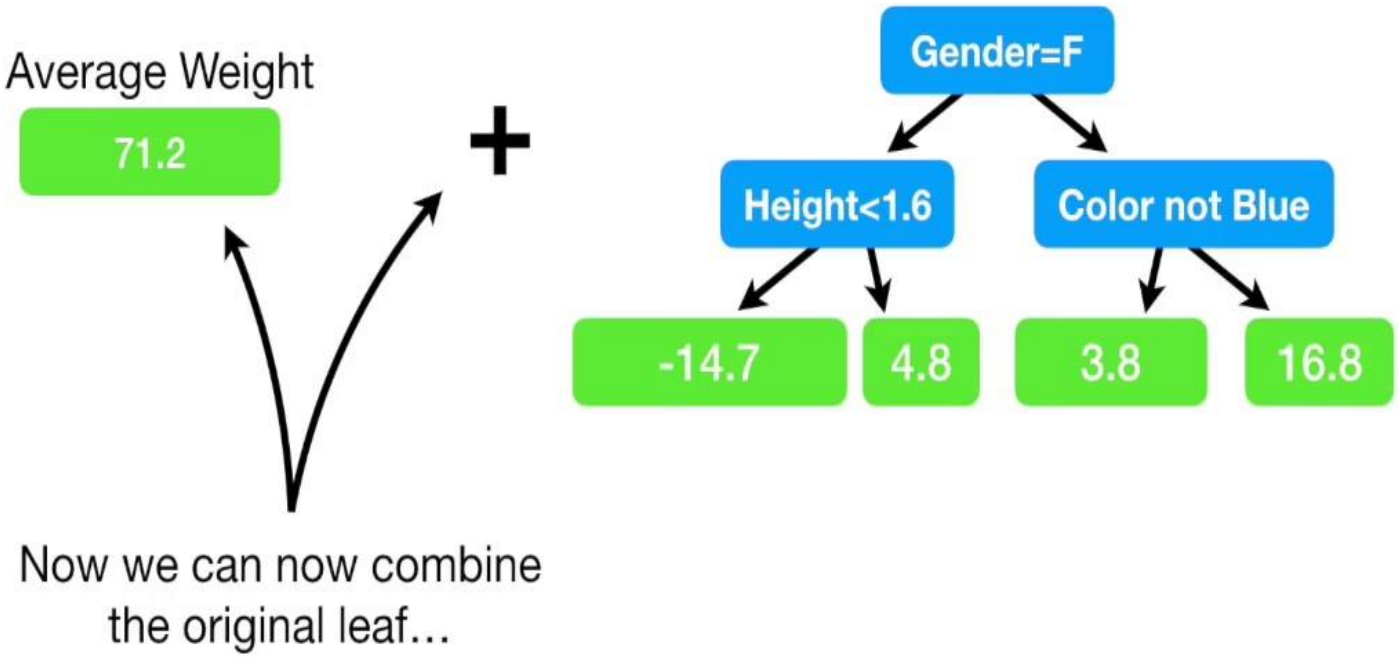
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



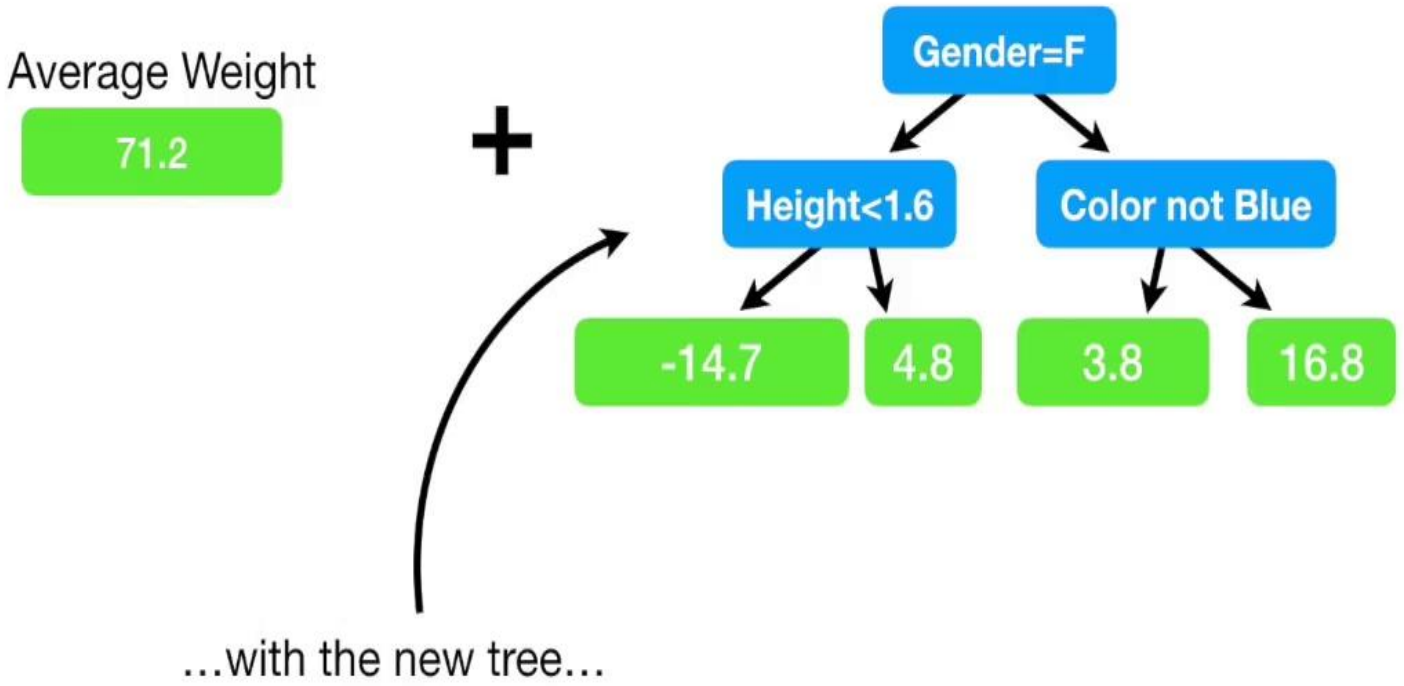
So we replace these residuals with their average.

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

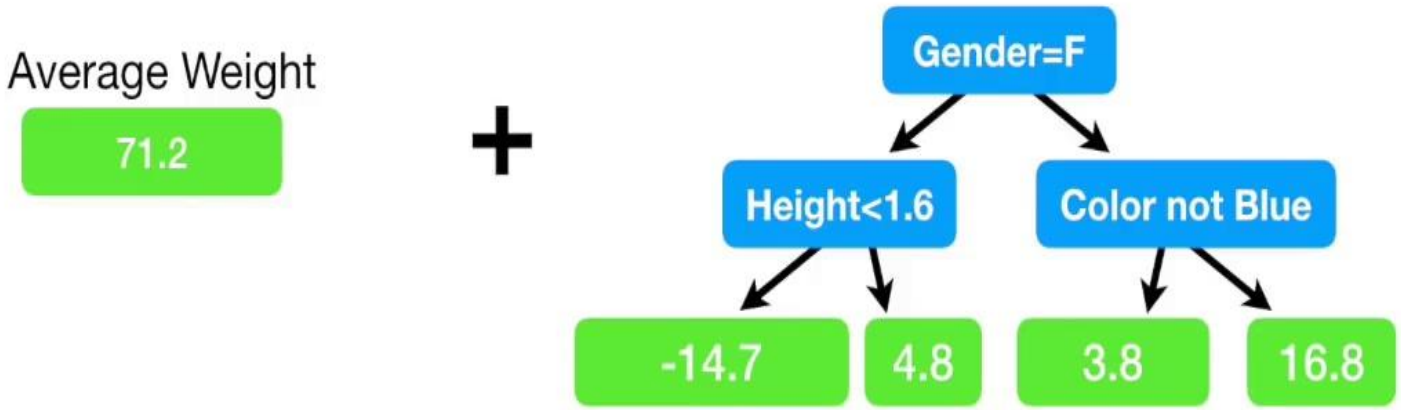
Example


Average Weight
71.2
+
Gender=F
Height<1.6
Color not Blue
-14.7
4.8
3.8
16.8
Now we can now combine
the original leaf…

Example


Average Weight
71.2
+
Gender=F
Height<1.6
Color not Blue
-14.7
4.8
3.8
16.8
...with the new tree...

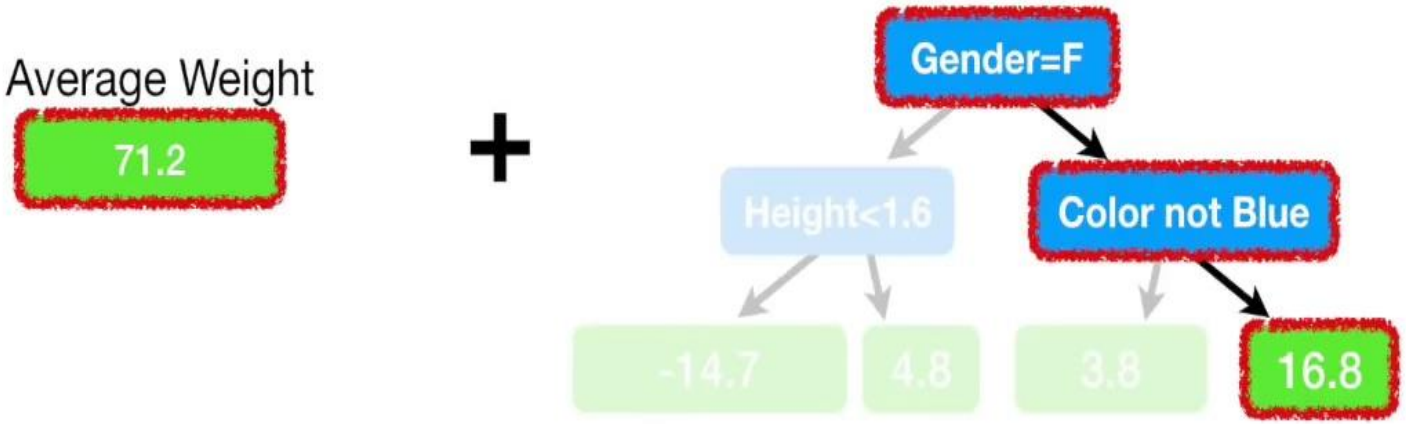
Example



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**.

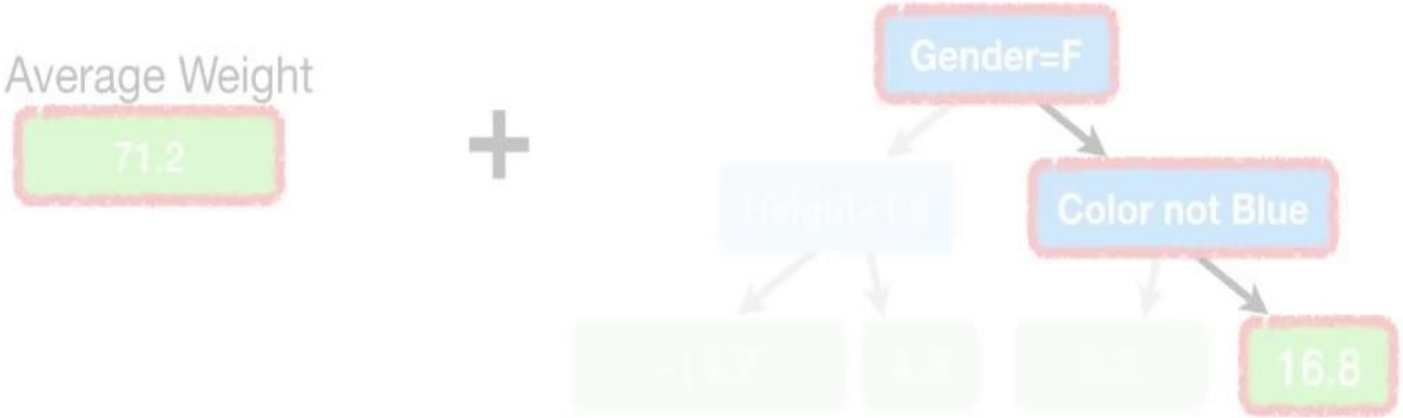
Example



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

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

Example

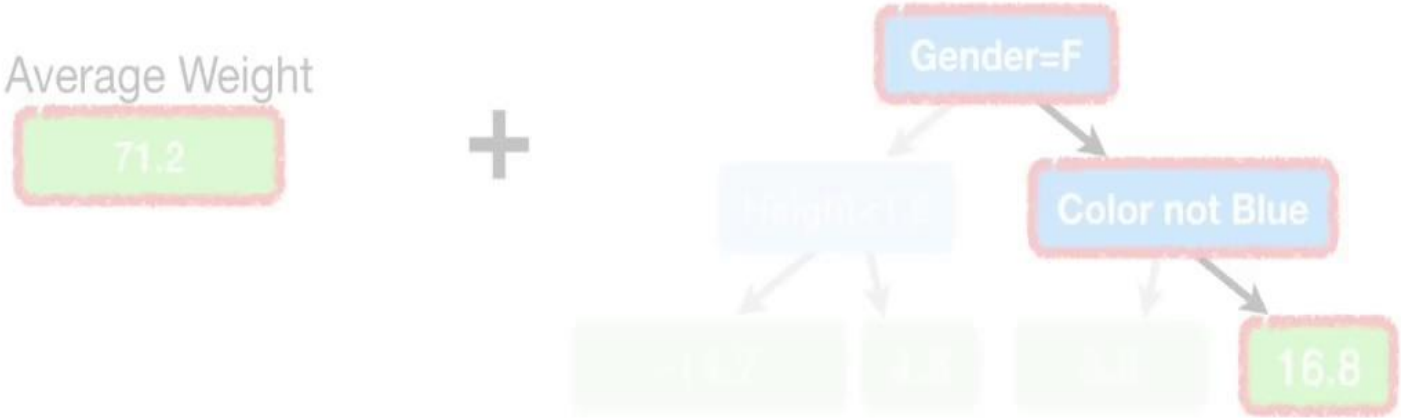


Predicted Weight = 71.2 + 16.8 = 88

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

Is this awesome???

Example

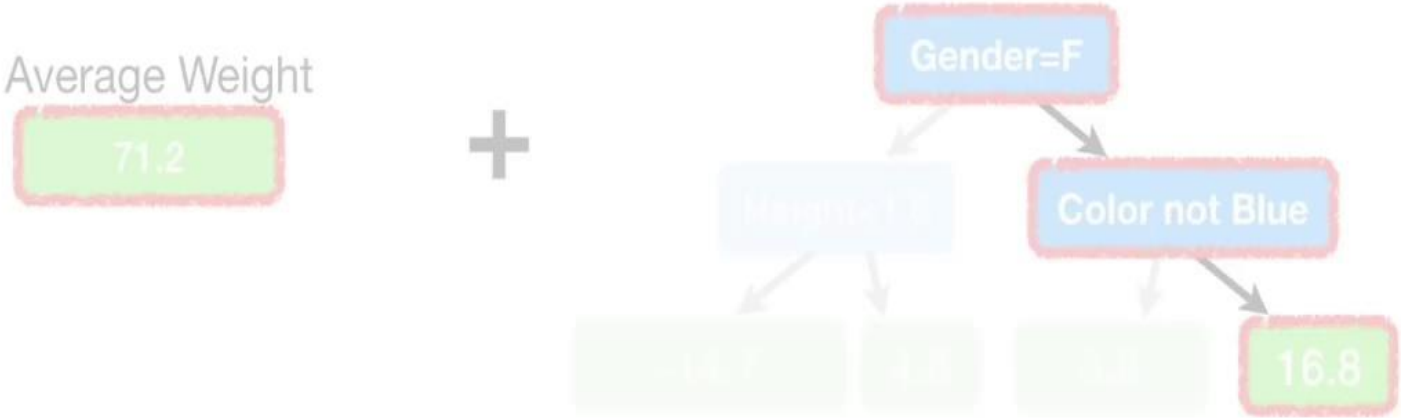


Predicted Weight = 71.2 + 16.8 = 88

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

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

Example

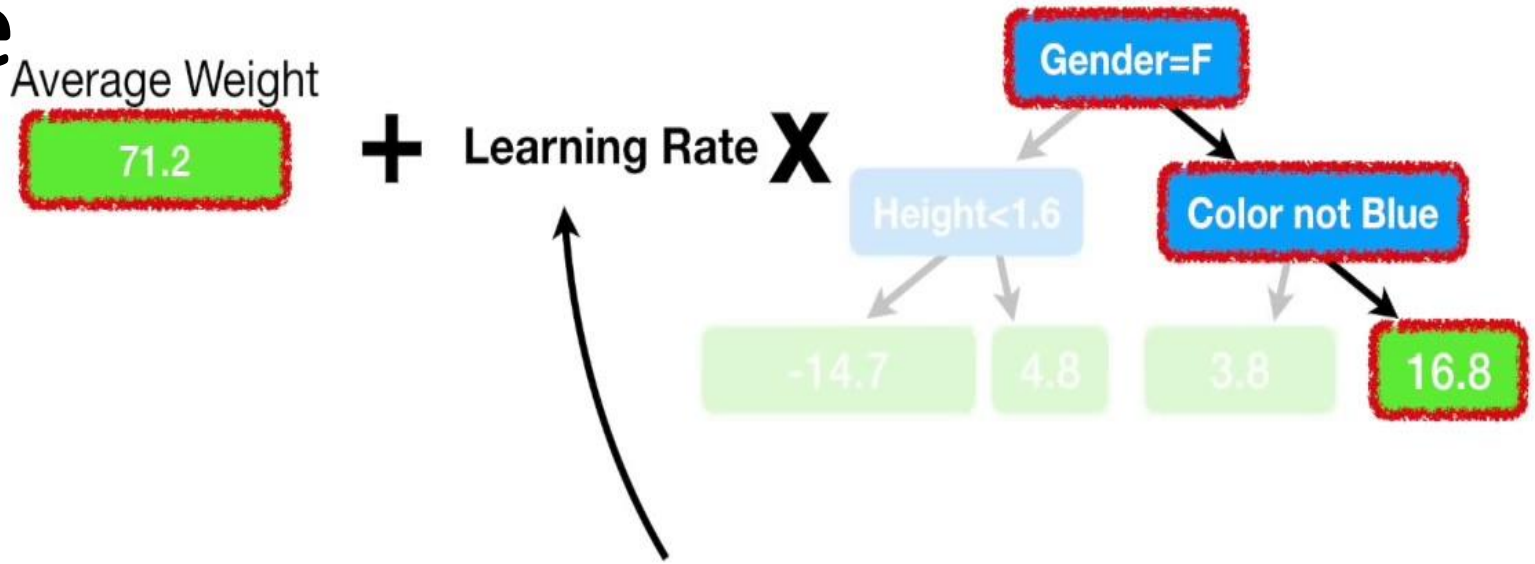


Predicted Weight = 71.2 + 16.8 = 88

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

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

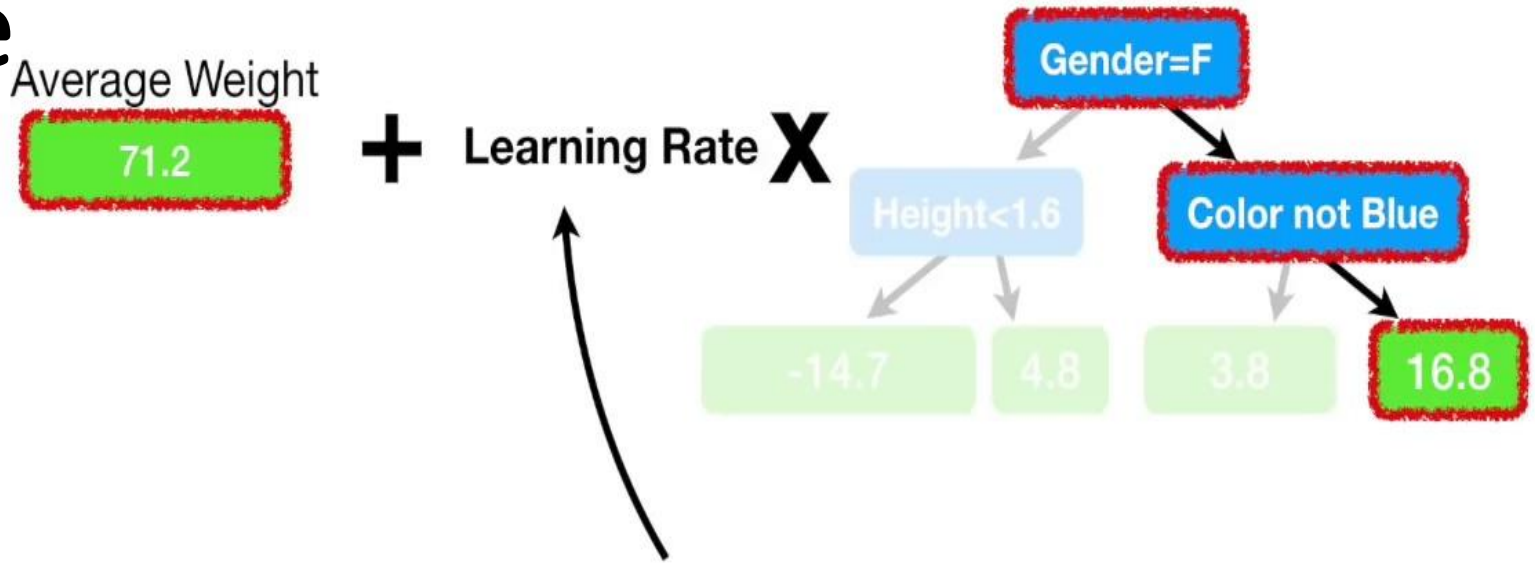
Example



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.

Example

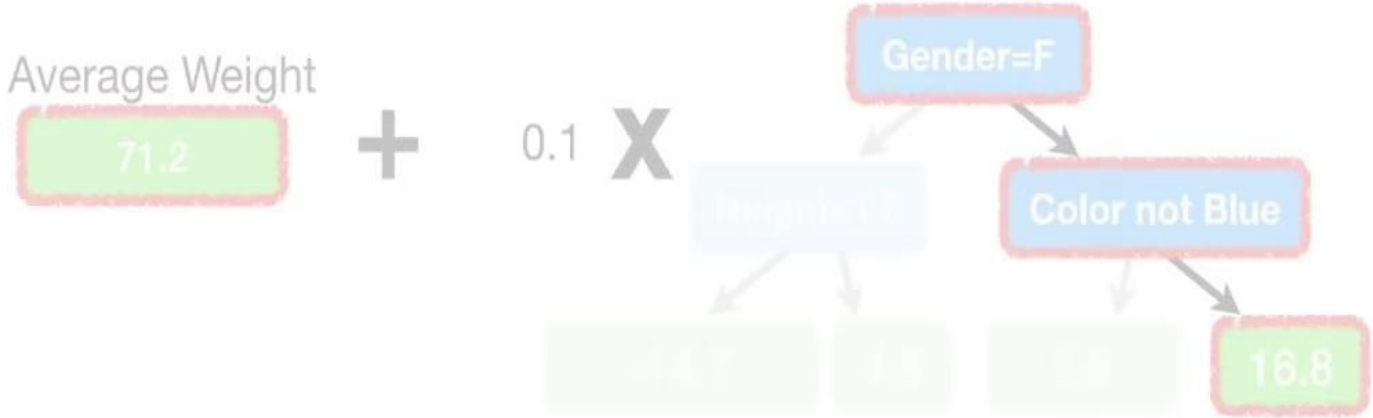


Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

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**.

Example

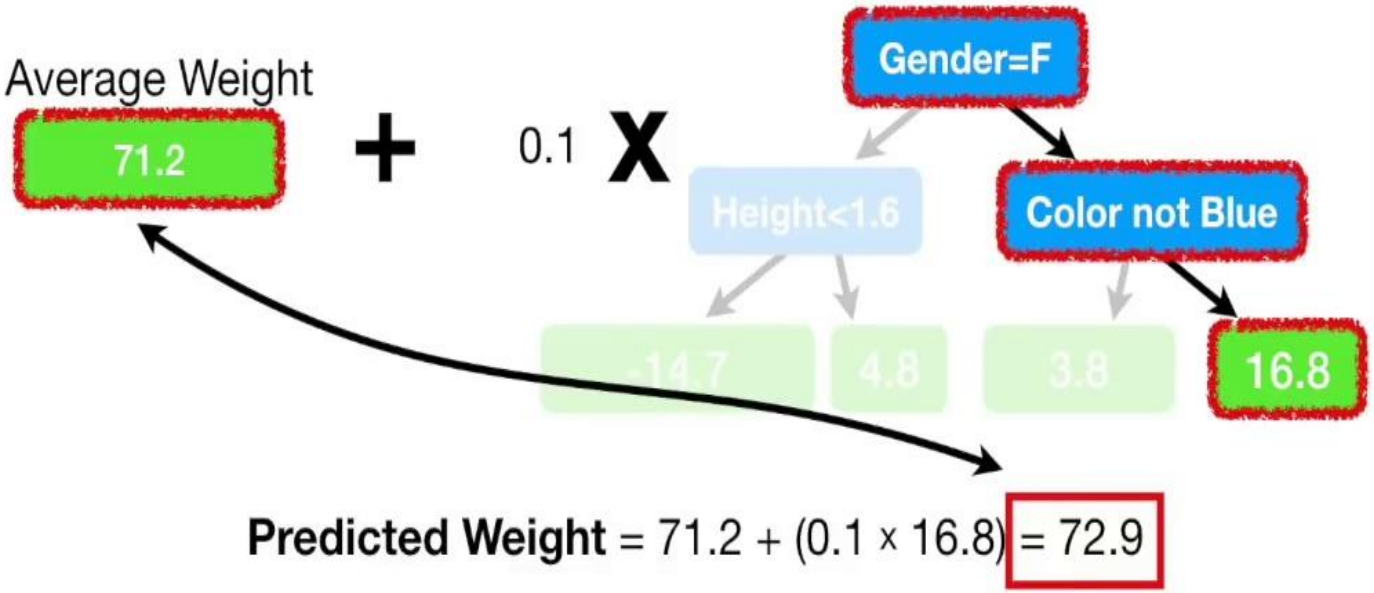


Predicted Weight = 71.2 + (0.1 x 16.8) = 72.9

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

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

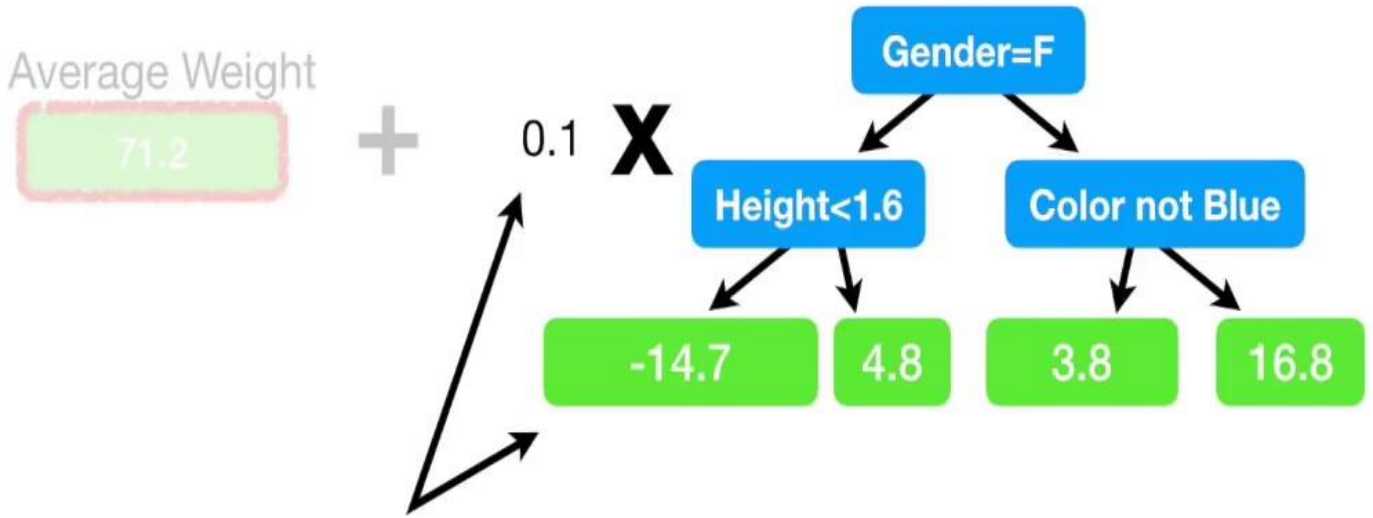
Example



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

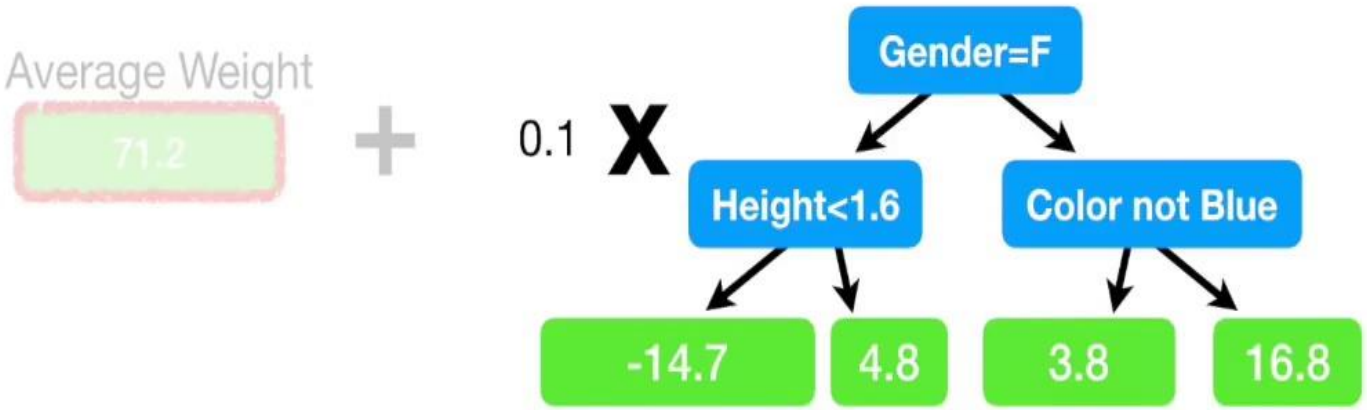
…but it's a little bit better than the **Prediction** made with just the original leaf, which predicted that all samples would weigh **71.2**.

Example



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

Example



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

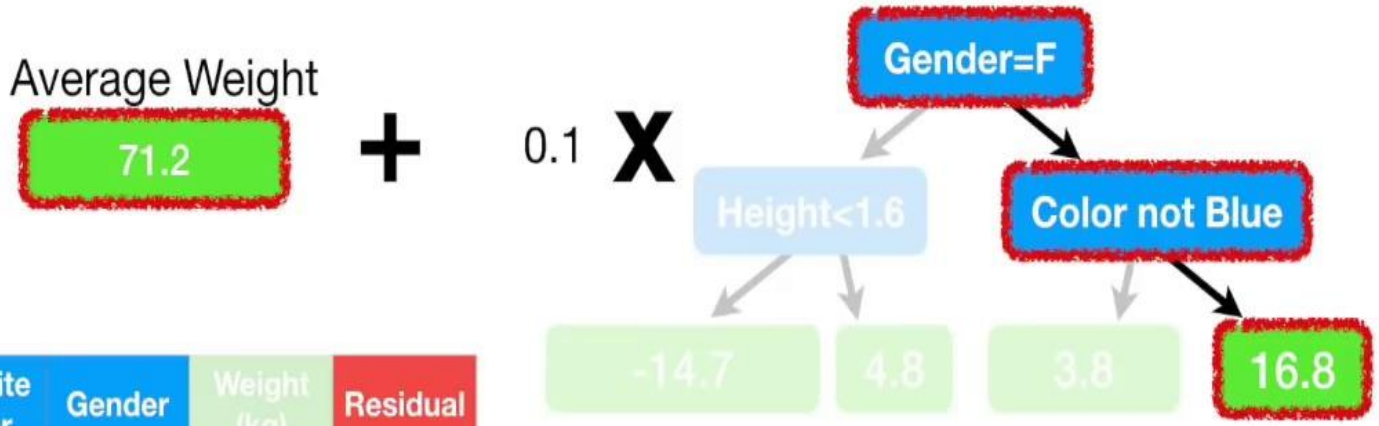
Example

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	

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

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

Example



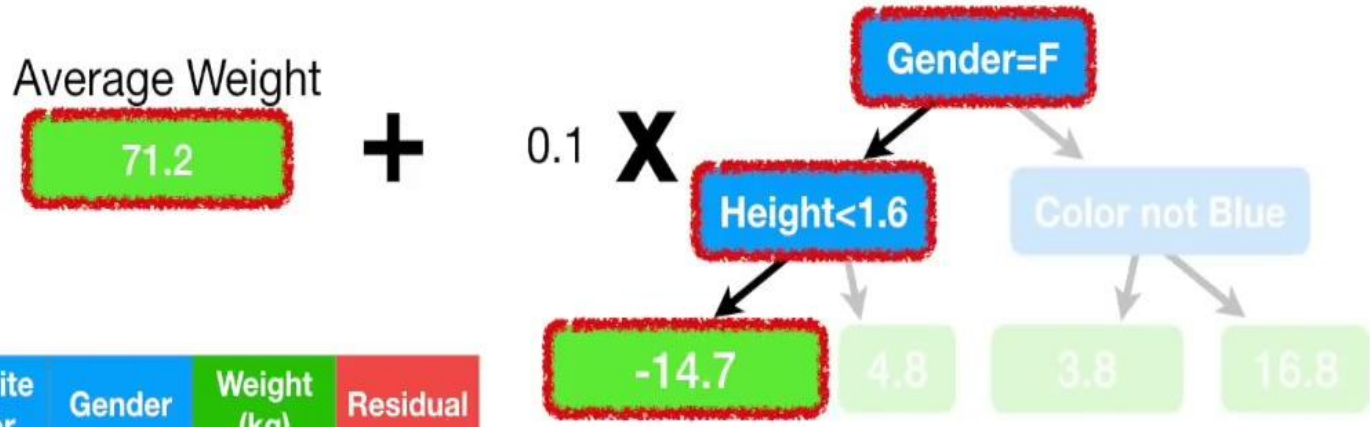
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	
1.6	Green	Female	76	
1.5	Blue	Female	56	
1.8	Red	Male	73	
1.5	Green	Male	77	
1.4	Blue	Female	57	

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

= 15.1

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

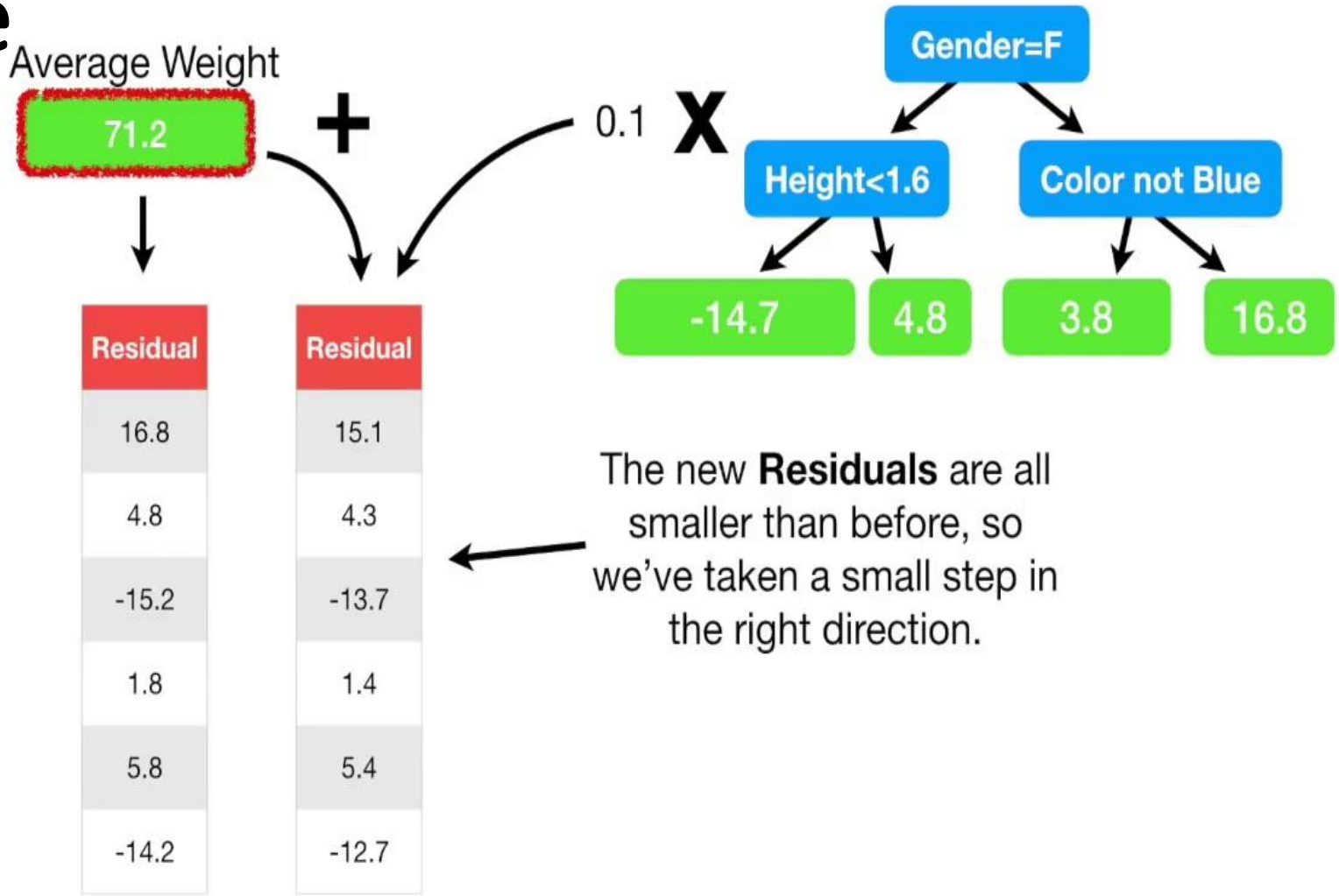
Example



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

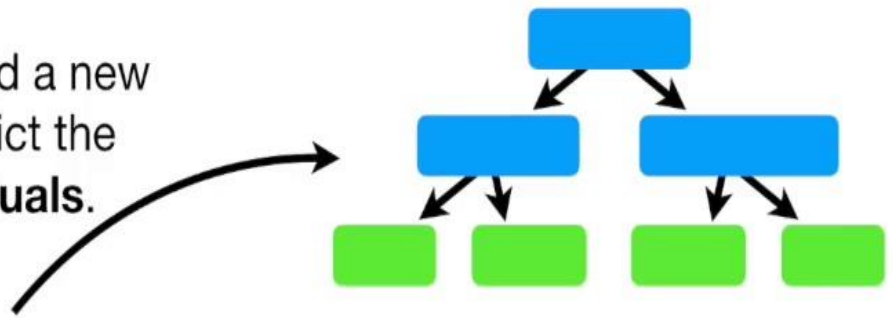
Residual = (Observed - Predicted)

Example


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.

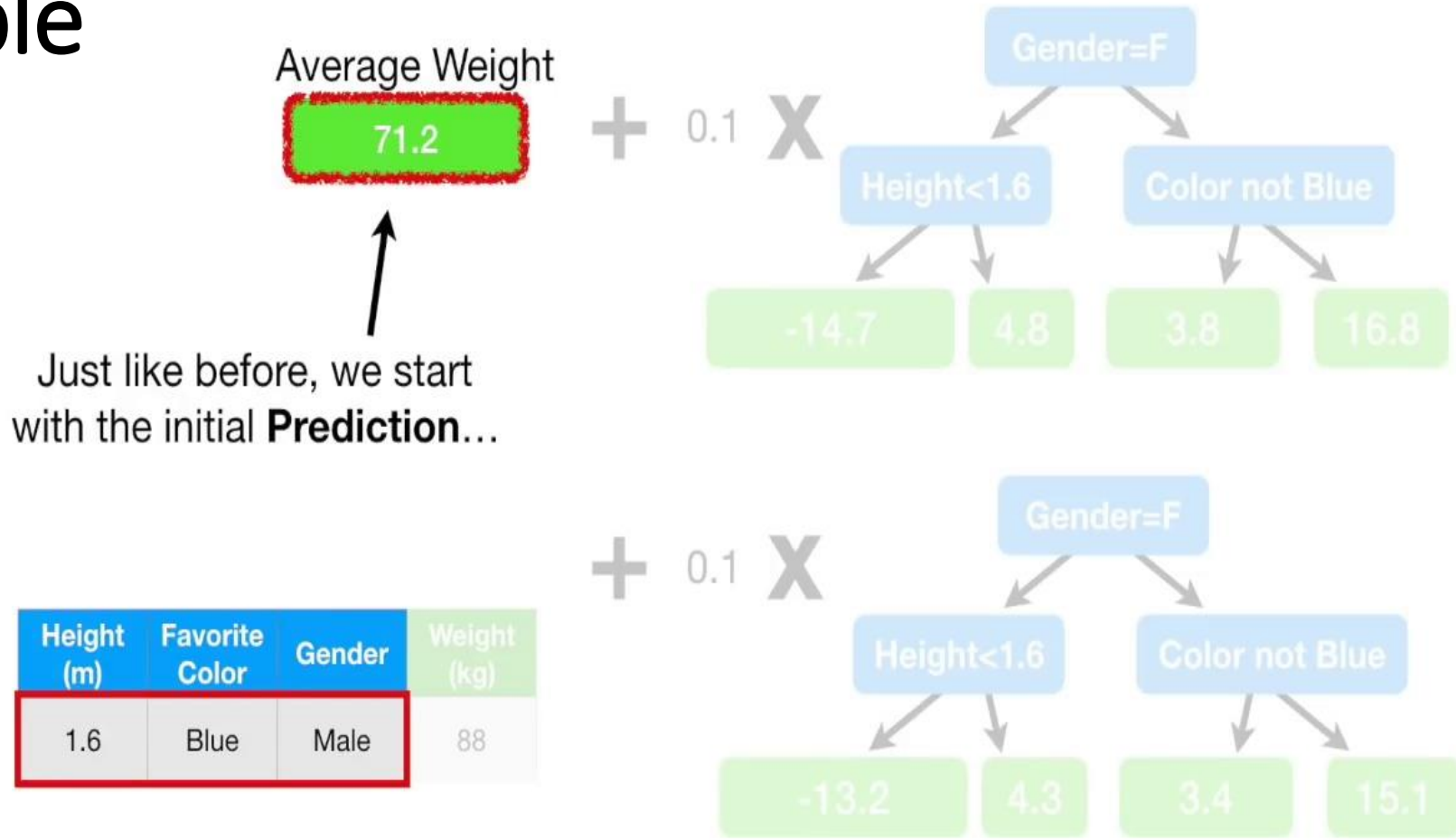
Example

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



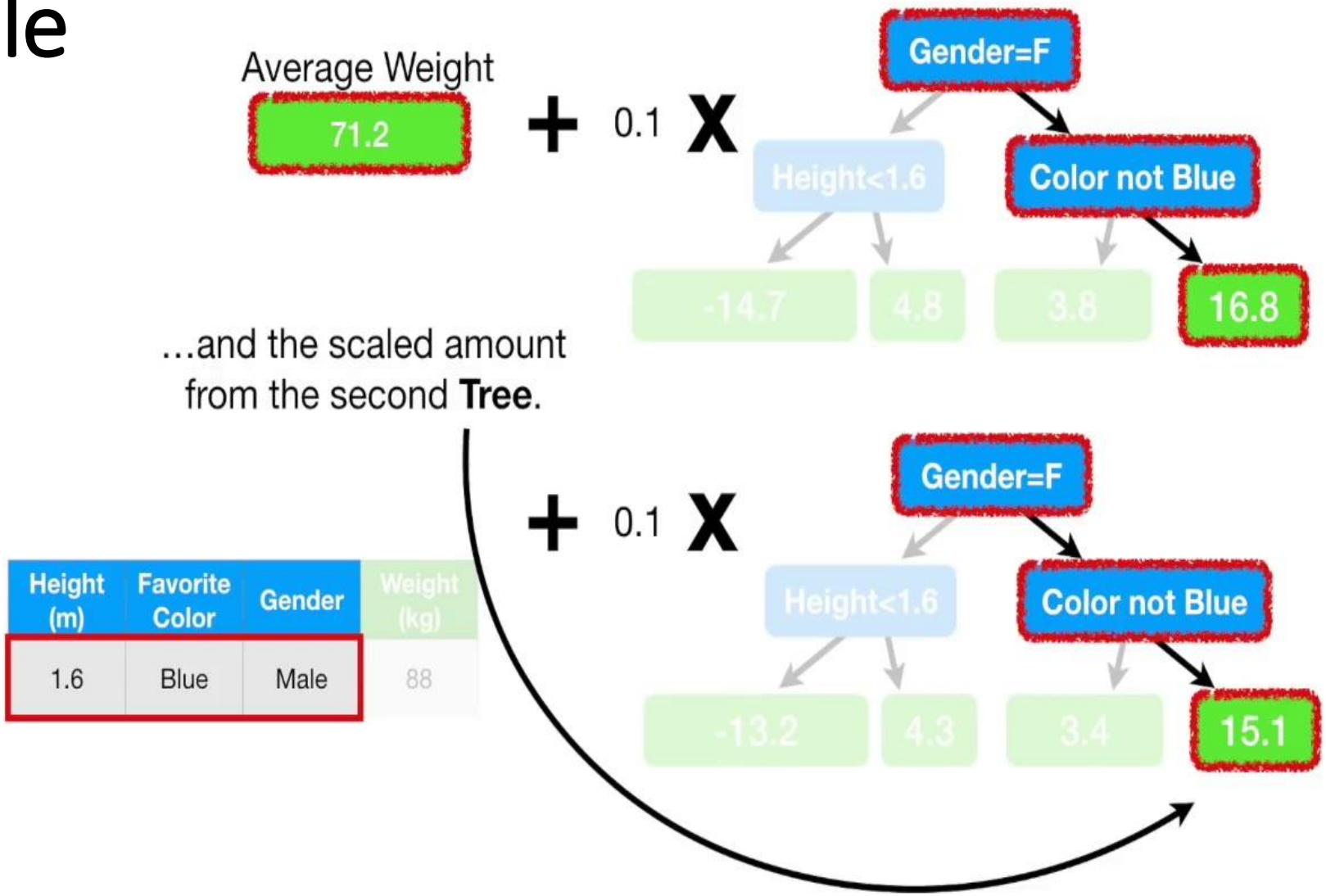
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

Example



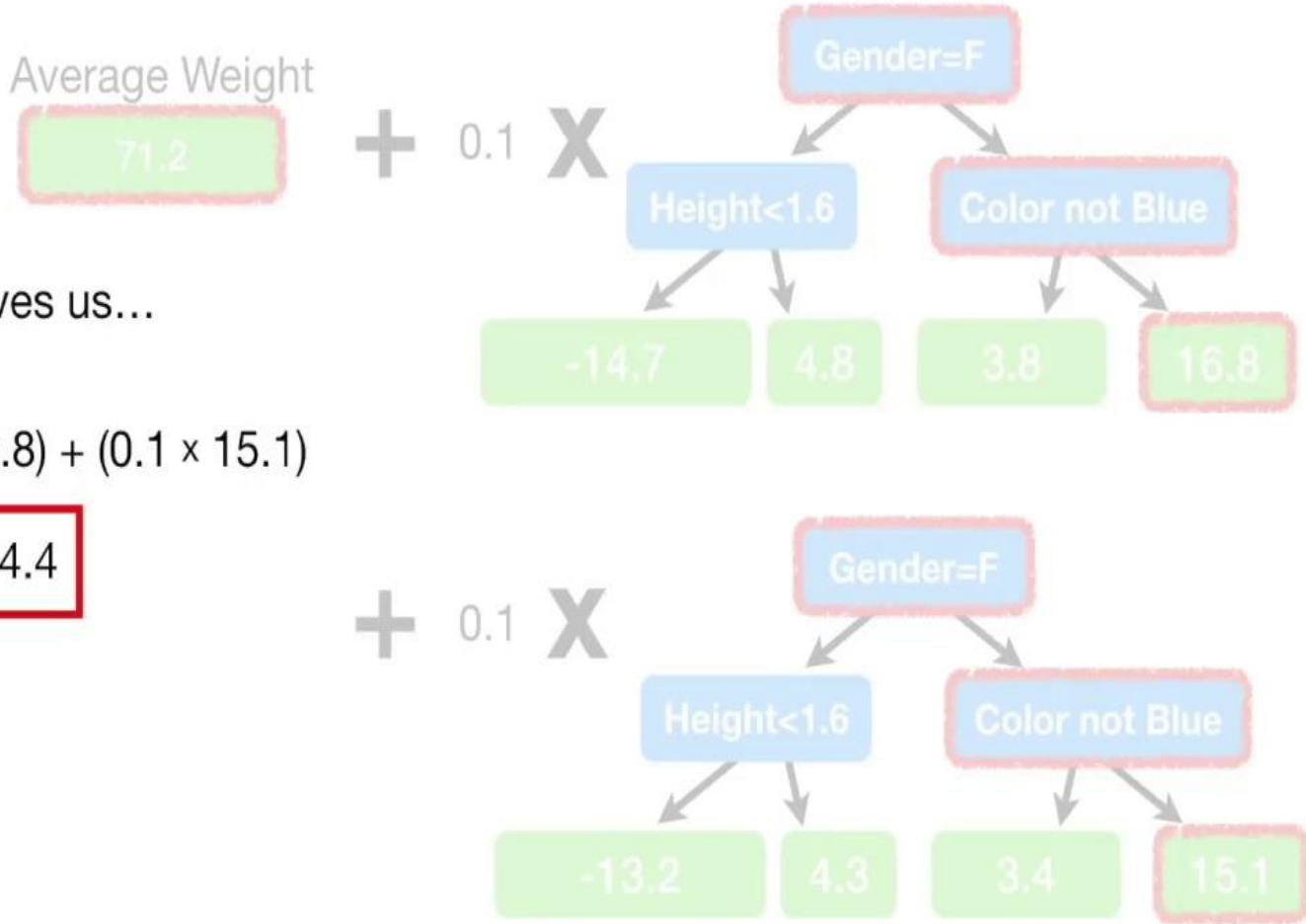
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

Example



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

Example

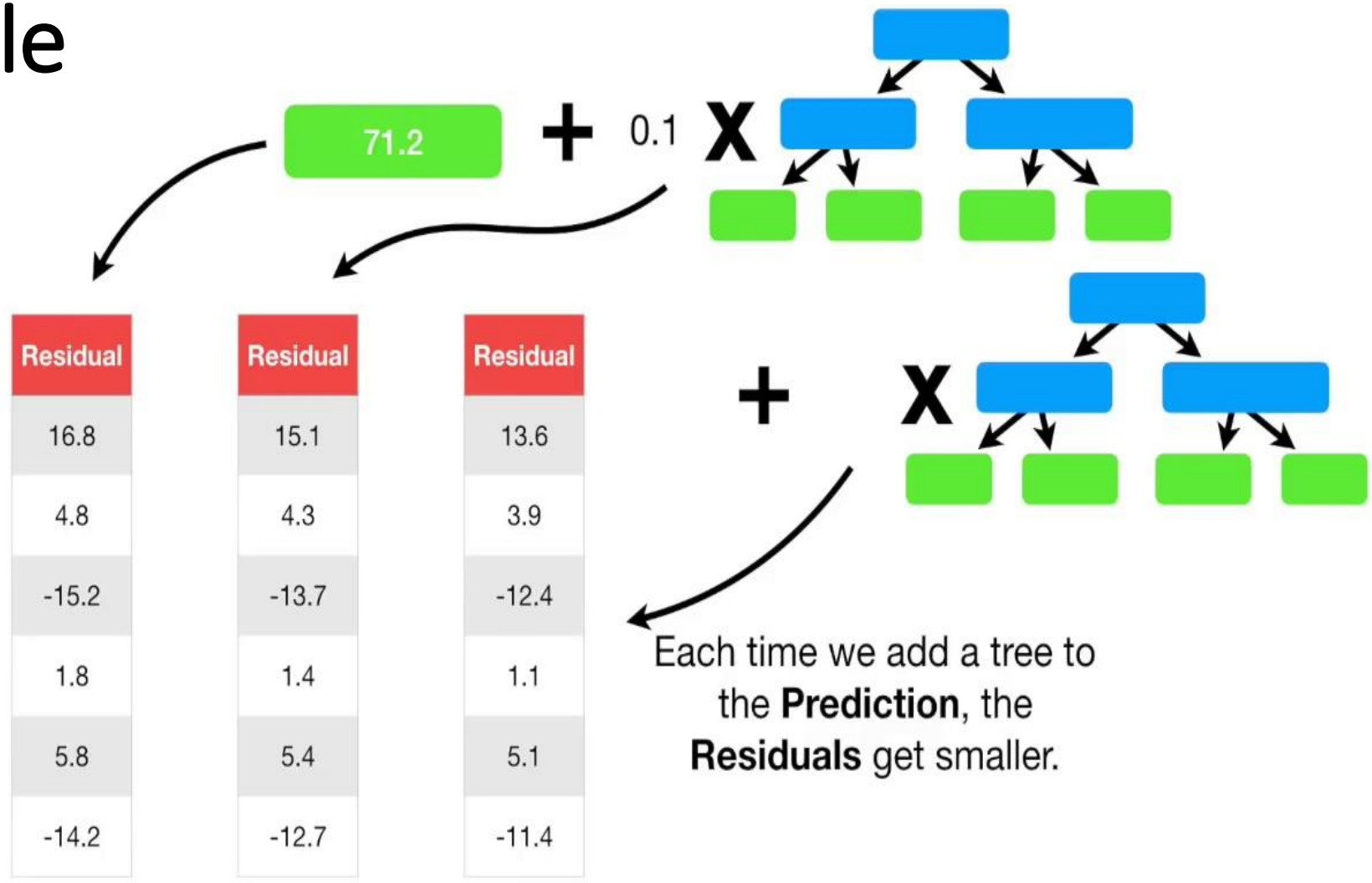


That gives us…

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

$= 74.4$

Example


71.2
+
0.1
X
+
X
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Residual
15.1
4.3
-13.7
1.4
5.4
-12.7
Residual
13.6
3.9
-12.4
1.1
5.1
-11.4
Each time we add a tree to the **Prediction**, the **Residuals** get smaller.

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Step 3: Output $F_M(x)$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$ and a differentiable **Loss Function** $L(y_i, F(x))$

The **Loss Function** that is most commonly used when doing **Regression** with **Gradient Boost** is…

$$\frac{1}{2}\,(\textbf{Observed} - \textbf{Predicted})^2$$

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

GB Algorithm

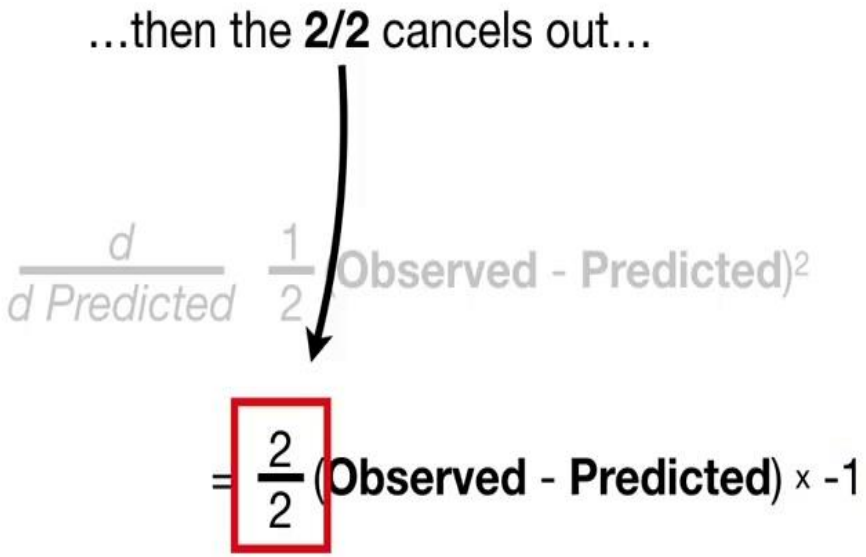
Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

The reason why people choose this **Loss Function** for **Gradient Boost** is that when we differentiate it with respect to "**Predicted**"…

$$\frac{d}{d\,Predicted}\ \frac{1}{2}(\textbf{Observed} - \textbf{Predicted})^2$$

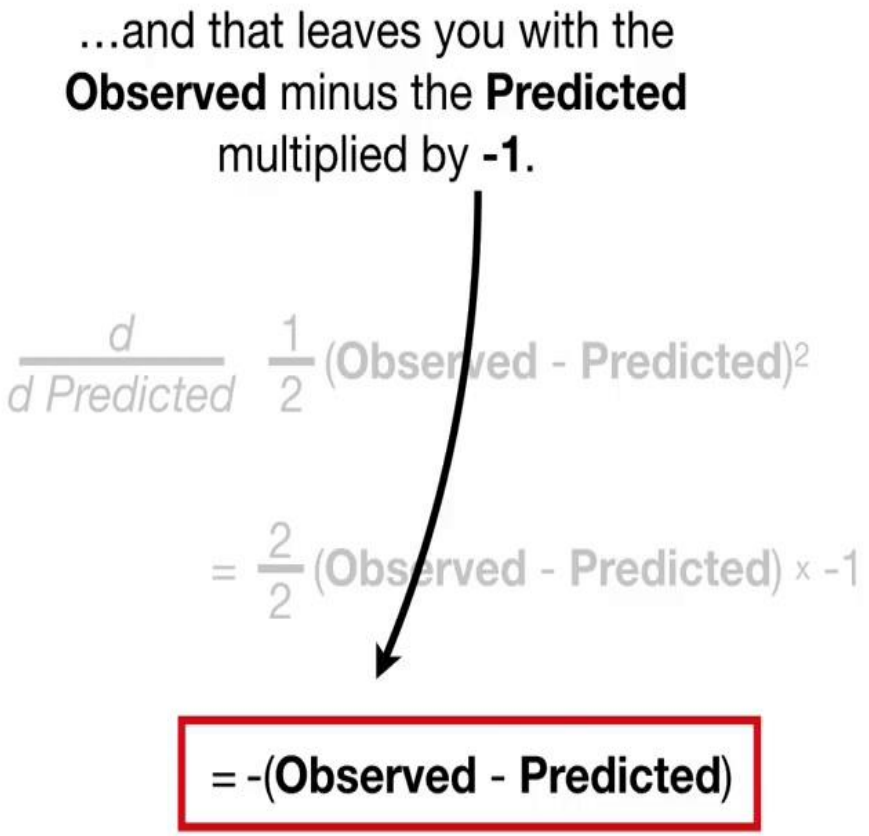
GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$



GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$



GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

In other words, we are left with the *negative* **Residual**, and this makes the math easier since **Gradient Boost** uses the derivative a lot.

$$\frac{d}{d\,Predicted}\ \frac{1}{2}(Observed - Predicted)^2$$

$$= \frac{2}{2}(Observed - Predicted) \times -1$$

$$= -(Observed - Predicted)$$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

The y_i's are the **Observed** values...

$\frac{1}{2}$(Observed - Predicted)^2

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

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y, F(x))$

...and ***F(x)*** is a function that gives us the **Predicted** values.

$$\frac{1}{2}(\text{Observed} - \text{Predicted})^2$$

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

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

We start by initializing the model with a constant value...

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\frac{1}{2}(88 - \textbf{Predicted})^2 +$$

$$\frac{1}{2}(76 - \textbf{Predicted})^2 +$$

$$\frac{1}{2}(56 - \textbf{Predicted})^2$$

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

The summation means that we add up one **Loss Function** for each **Observed** value…

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$$\frac{1}{2}(88 - \textbf{Predicted})^2 + \frac{1}{2}(76 - \textbf{Predicted})^2 + \frac{1}{2}(56 - \textbf{Predicted})^2$$

…and the "**argmin over gamma**" means we need to find a **Predicted** value that minimizes this sum.

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

GB Algorithm

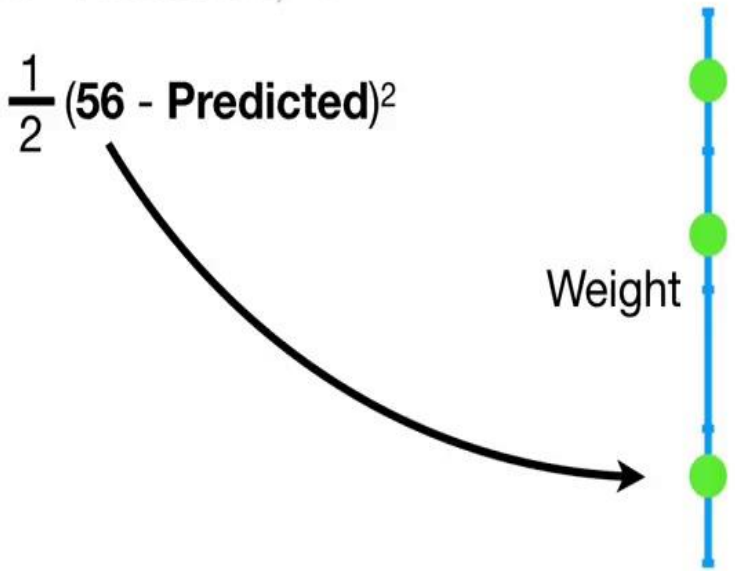
Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$\frac{1}{2}$ (88 - **Predicted**)^2 +

$\frac{1}{2}$ (76 - **Predicted**)^2 +

$\frac{1}{2}$ (56 - **Predicted**)^2



In other words, if we plot the **Observed Weights** on a number line…

GB Algorithm

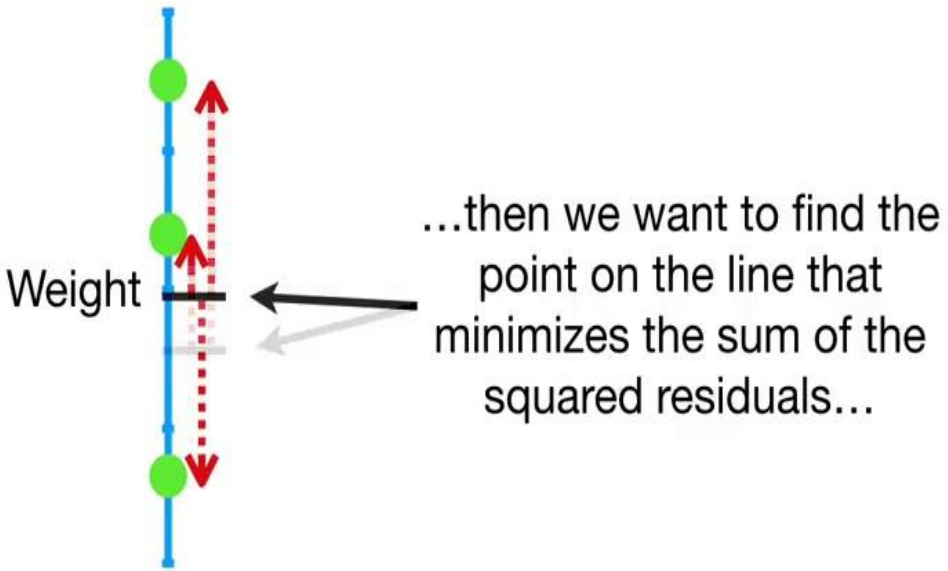
Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

$\frac{1}{2}$ (88 - **Predicted**)^2 +

$\frac{1}{2}$ (76 - **Predicted**)^2 +

$\frac{1}{2}$ (56 - **Predicted**)^2



GB Algorithm

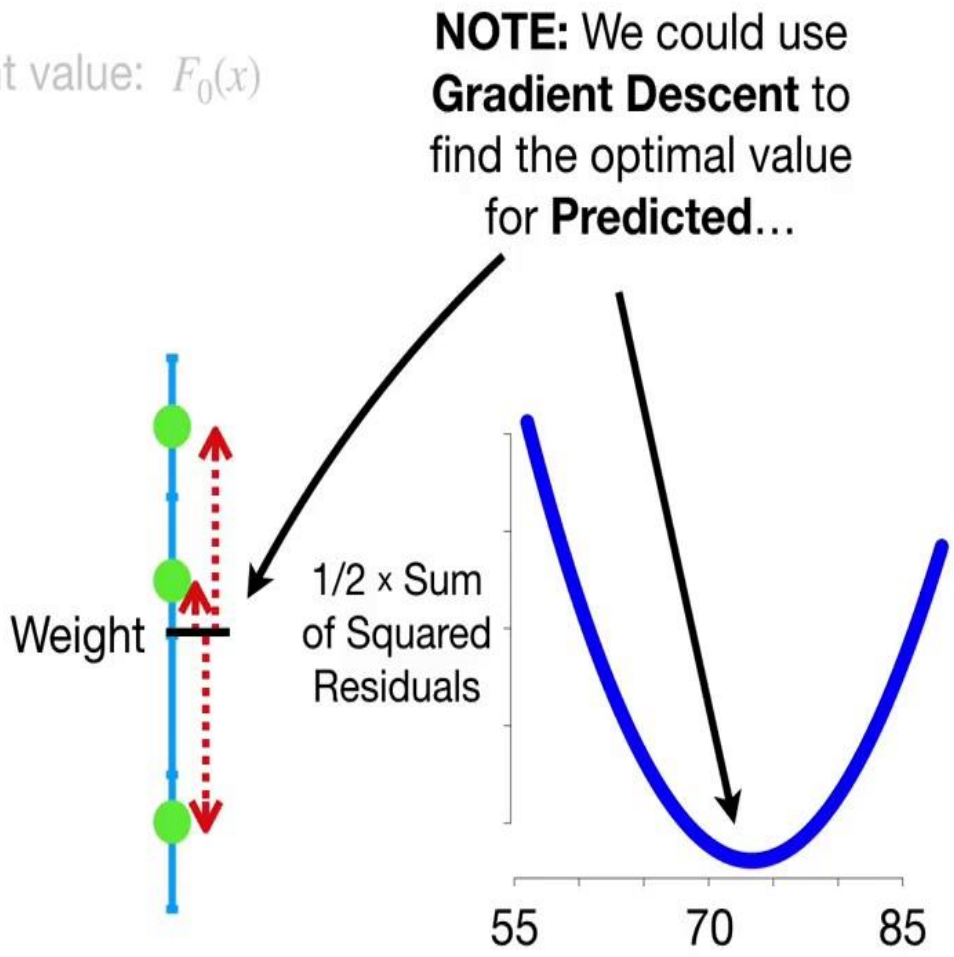
Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Fu**

Step 1: Initialize model with a constant value: $F_0(x)$

$$\frac{1}{2}(88 - \text{Predicted})^2 +$$

$$\frac{1}{2}(76 - \text{Predicted})^2 +$$

$$\frac{1}{2}(56 - \text{Predicted})^2$$



GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

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

Predicted $= \frac{88 + 76 + 56}{3}$

…and we end up with the **Average** of the **Observed Weights**.

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

We have now created the initial predicted value, $F_0(x)$…

$$F_0(x) = \frac{88 + 76 + 56}{3}$$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

…and it equals **73.3**.

$$F_0(x) = \frac{88 + 76 + 56}{3}$$

$$= 73.3$$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

That means that the initial predicted value, ***F₀(x)***, is just a leaf.



73.3

$$F_0(x) = \frac{88 + 76 + 56}{3}$$

$$= 73.3$$

GB Algorithm

Now we can work on **Step 2**…

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_m I(x \in R_{jm})$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

This part is just the derivative of the **Loss Function**…

$$\frac{d}{d\ Predicted}\ \frac{1}{2}\ (\textbf{Observed - Predicted})^2$$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

This part is just the derivative of the **Loss Function**…

$$\frac{d}{d\,Predicted}\frac{1}{2}(\textbf{Observed} - \textbf{Predicted})^2$$

…with respect to the **Predicted** value…

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

$$\frac{d}{d\,Predicted}\frac{1}{2}(\textbf{Observed - Predicted})^2$$

…and we've already calculated this.

$$= -(\textbf{Observed - Predicted})$$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

Height (m)	Favorite Color	Gender	Weight (kg)	$r_{i,1}$
1.6	Blue	Male	88	14.7
1.6	Green	Female	76	2.7
1.5	Blue	Female	56	-17.3

We've finished **Part A** of **Step 2** by calculating a **Residual** for each sample.

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

NOTE: Before we move on, I just want to point out that this derivative is the **Gradient** that **Gradient Boost** is named after.

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

I also want to point that the $r_{i,m}$ values are technically called **Pseudo Residuals**.

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

All this is saying is that we will build a regression tree…

GB Algorithm

Now let's do **Part C.**

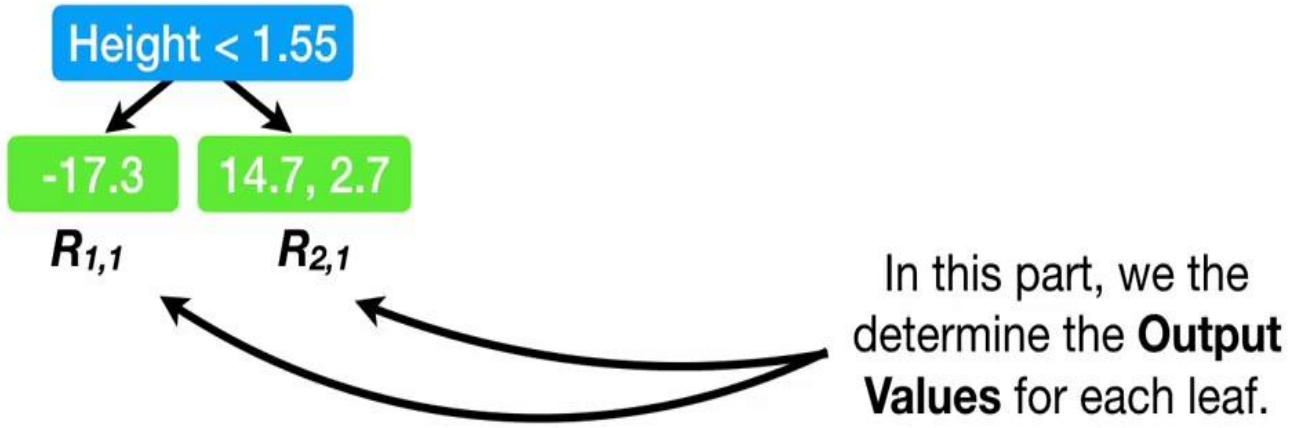
Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

GB Algorithm



(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

GB Algorithm

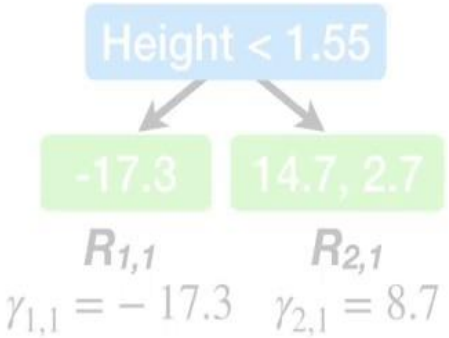
Step 1: Initialize model with a constant value: $$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

NOTE: This minimization is like what we did in **Step 1**.



(C) For $j = 1 \ldots J_m$ compute $$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

GB Algorithm



Given our choice of **Loss Function**, the **Output Values** are *always* the average of the **Residuals** that end up in the same leaf.

$$\frac{1}{2}(\textbf{Observed} - \textbf{Predicted})^2$$

GB Algorithm

Now let's do **Part D**!!!

Step 2: for $m = 1$ to M:

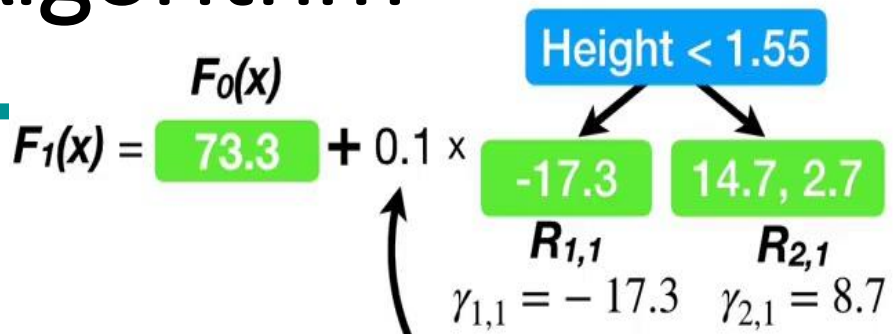
(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

GB Algorithm



In this example, we'll set **nu** to **0.1**.

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

GB Algorithm



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

Now we will use ***F₁(x)*** to make new **Predictions** for each sample.

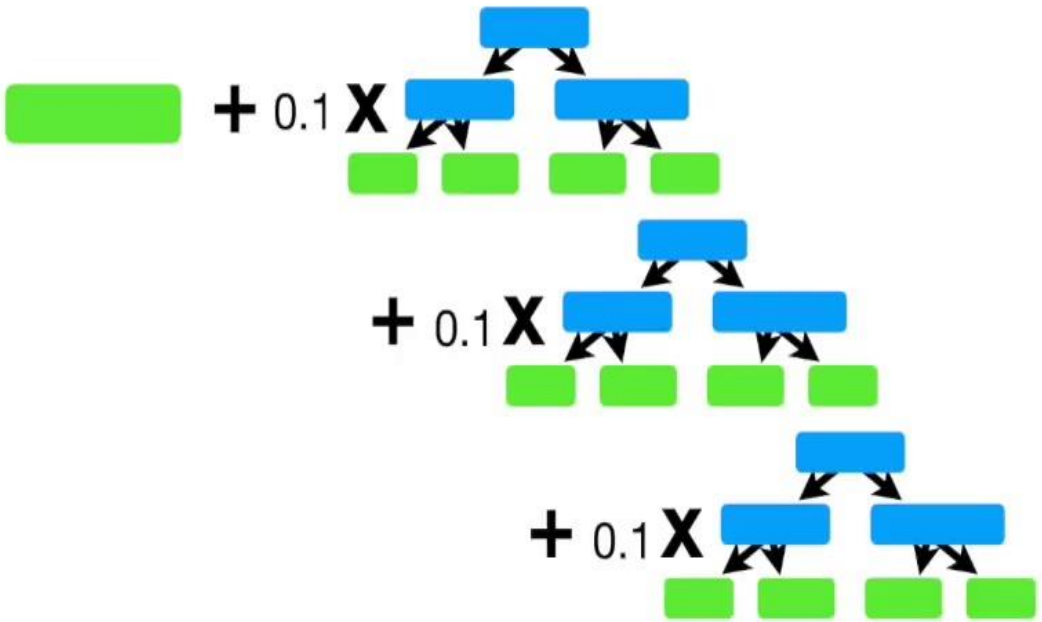
(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Gradient Boosting for Classification

Example

...and walk through, step-by-step, the most common way that **Gradient Boost** fits a model to this **Training Data**.

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes



Example



When we use **Gradient Boost for Classification**, the initial **Prediction** for every individual is the **log(odds)**.

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Example

log(4/2) = 0.7 ← So this is the **Initial Prediction**.

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Example

log(4/2) = 0.693

Just like with **Logistic Regression**, the easiest way to use the **log(odds)** for **Classification** is to convert it to a **Probability**...

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Example

log(4/2) = 0.693

Just like with **Logistic Regression**, the easiest way to use the **log(odds)** for **Classification** is to convert it to a **Probability**...

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

...and we do that with a **Logistic Function**.

Example

log(4/2) = 0.693

So we plug the **log(odds)** into the **Logistic Function**...

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Probability of Loving Troll 2 $= \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$

Example

log(4/2) = 0.693

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…and we get **0.7** as the **Probability of Loving Troll 2.**

$$\text{Probability of Loving Troll 2} = \frac{e^{\log(4/2)}}{1 + e^{\log(4/2)}} = 0.7$$

0.666

Example

log(4/2) = 0.693

NOTE this are rounded values

...and we get **0.7** as the **Probability of Loving Troll 2.**

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

$$\text{Probability of Loving Troll 2} = \frac{e^{\log(4/2)}}{1 + e^{\log(4/2)}} = 0.7$$

0.666

Example

log(4/2) = 0.7

Probability of **Loving Troll 2** = 0.7

0.666

Since the **Probability** of **Loving Troll 2** is greater than **0.5**, we can **Classify** everyone in the **Training Dataset** as someone who **Loves Troll 2**.

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Example

log(4/2) = 0.693

Probability of **Loving Troll 2** = 0.7 0.666

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

We can measure how bad the initial **Prediction** is by calculating **Pseudo Residuals**, the difference between the **Observed** and the **Predicted** values.

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

Example

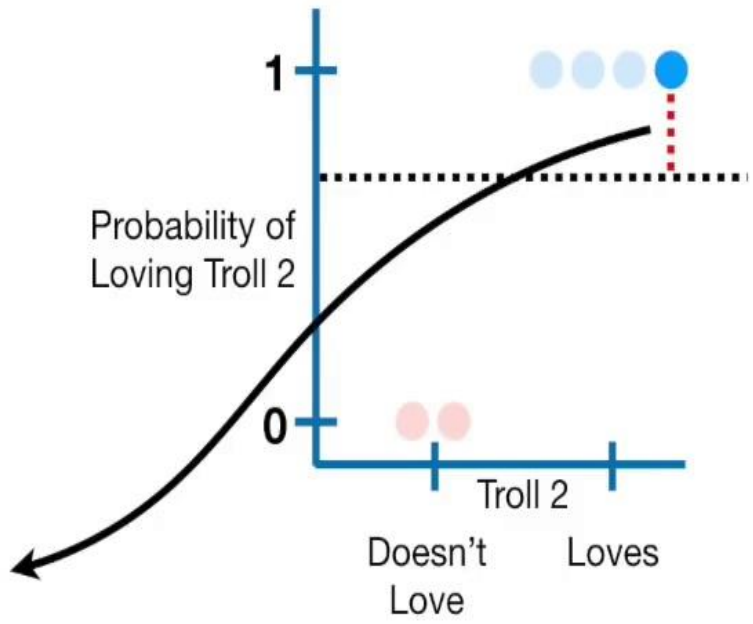
log(4/2) = 0.7

Probability of Loving Troll 2 = 0.7

Then we calculate the rest of the **Residuals**…

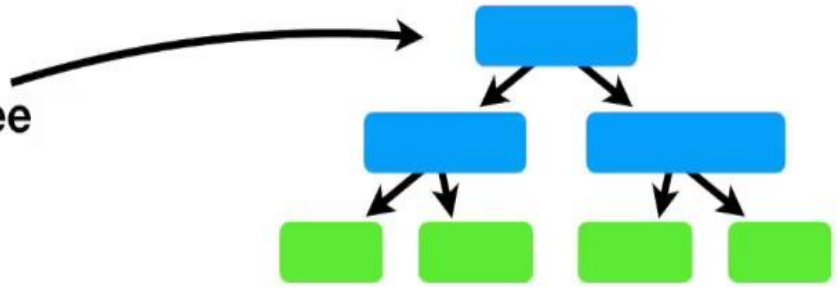
Residual = (**Observed** - **Predicted**)

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3



Example

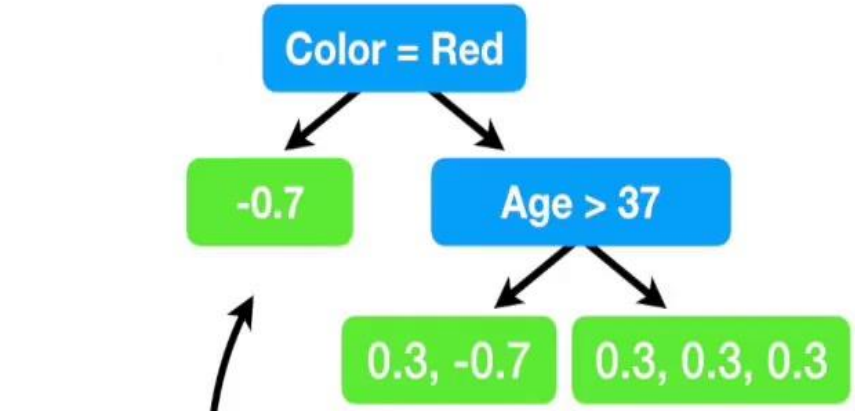
Now we will build a **Tree**



Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3

Example

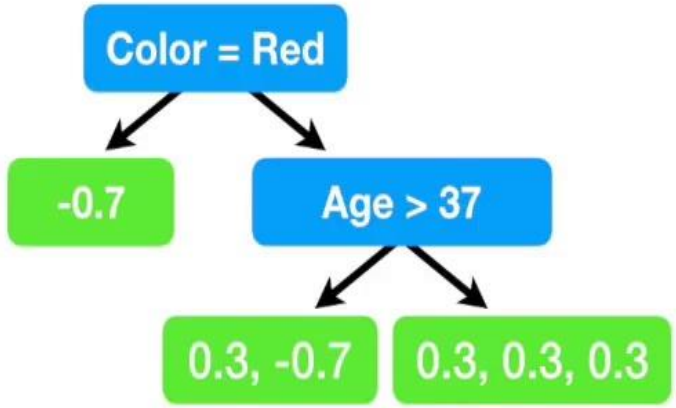
Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3



And here's the tree!

Example

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Residual
Yes	12	Blue	Yes	0.3
Yes	87	Green	Yes	0.3
No	44	Blue	No	-0.7
Yes	19	Red	No	-0.7
No	32	Green	Yes	0.3
No	14	Blue	Yes	0.3



In this simple example, we are limiting the number of leaves to **3**.

In practice people often set the maximum number of leaves to be between **8** and **32**

Example



When we used **Gradient Boost** for **Regression**, a leaf with single **Residual** had an **Output Value** equal to that **Residual**.

Example



In contrast, when we use **Gradient Boost** for **Classification**, the situation is a little more complex.

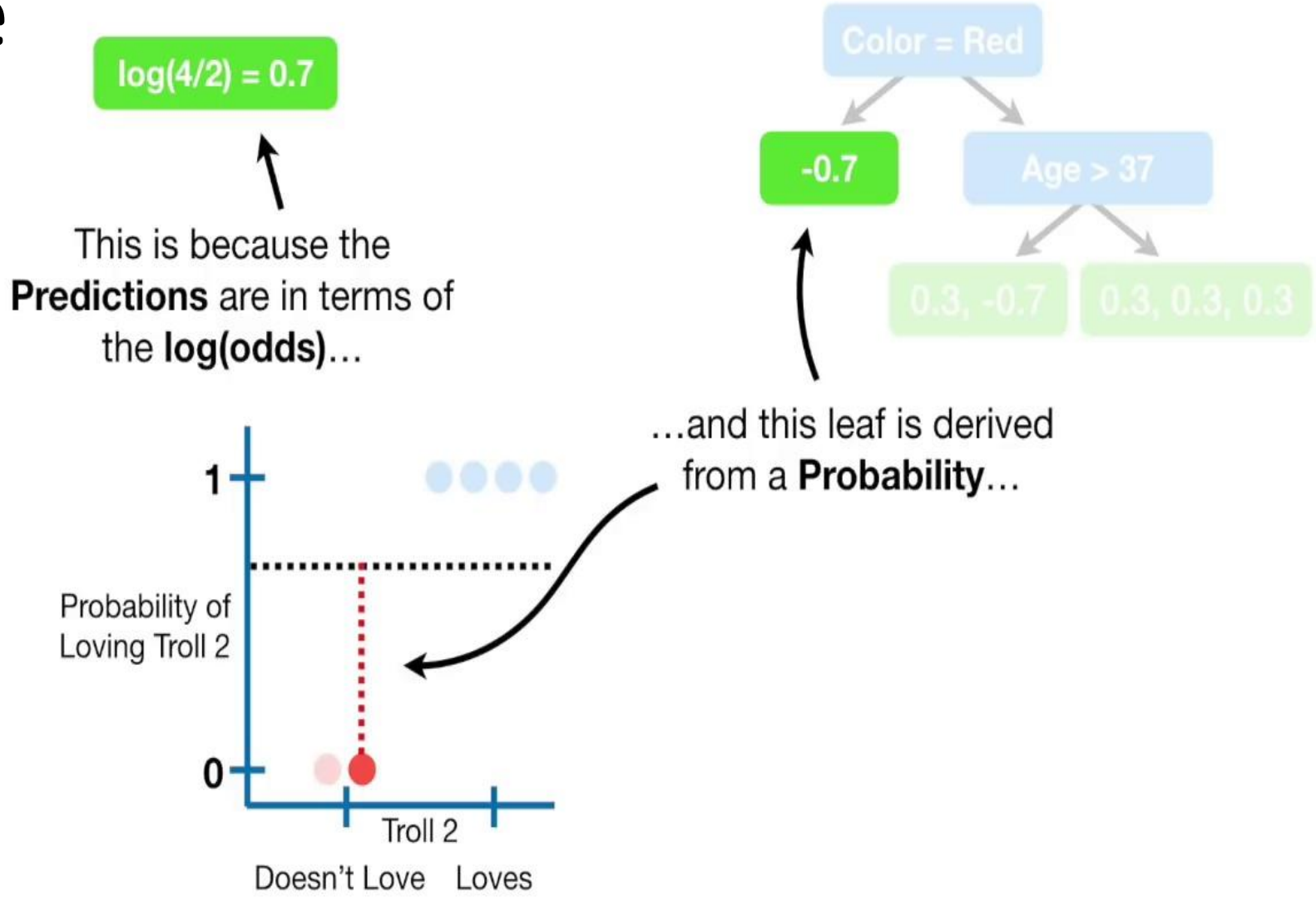
Example

log(4/2) = 0.7

This is because the **Predictions** are in terms of the **log(odds)**...



Example


log(4/2) = 0.7
This is because the **Predictions** are in terms of the **log(odds)**...
Color = Red
-0.7
Age > 37
0.3, -0.7
0.3, 0.3, 0.3
...and this leaf is derived from a **Probability**...
1
0
Probability of Loving Troll 2
Troll 2
Doesn't Love
Loves

Example


log(4/2) = 0.7
Color = Red
-0.7
Age > 37
0.3, -0.7
0.3, 0.3, 0.3
...so we can't just add them together to get a new **log(odds) Prediction** without some sort of transformation.

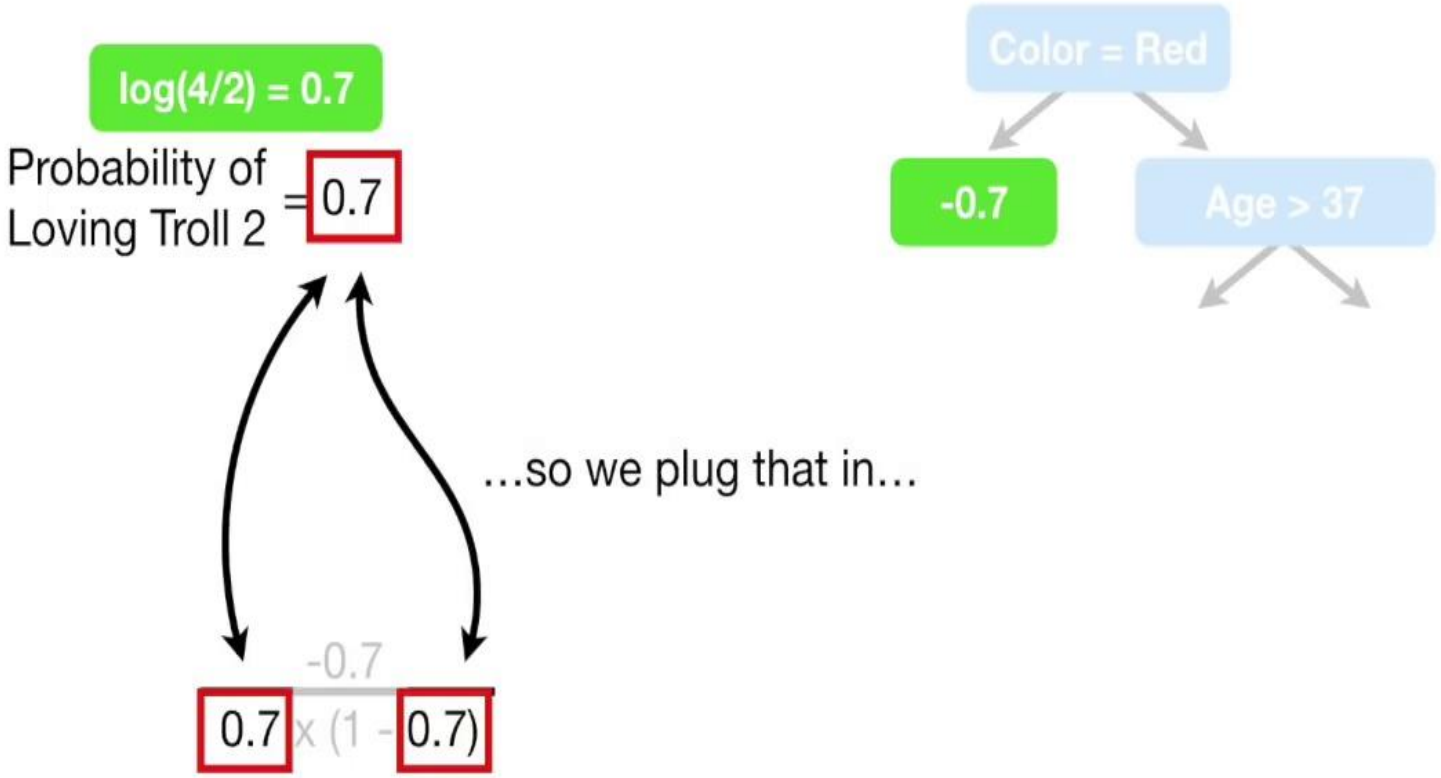
Example

When we use **Gradient Boost** for **Classification**, the most common transformation is the following formula.

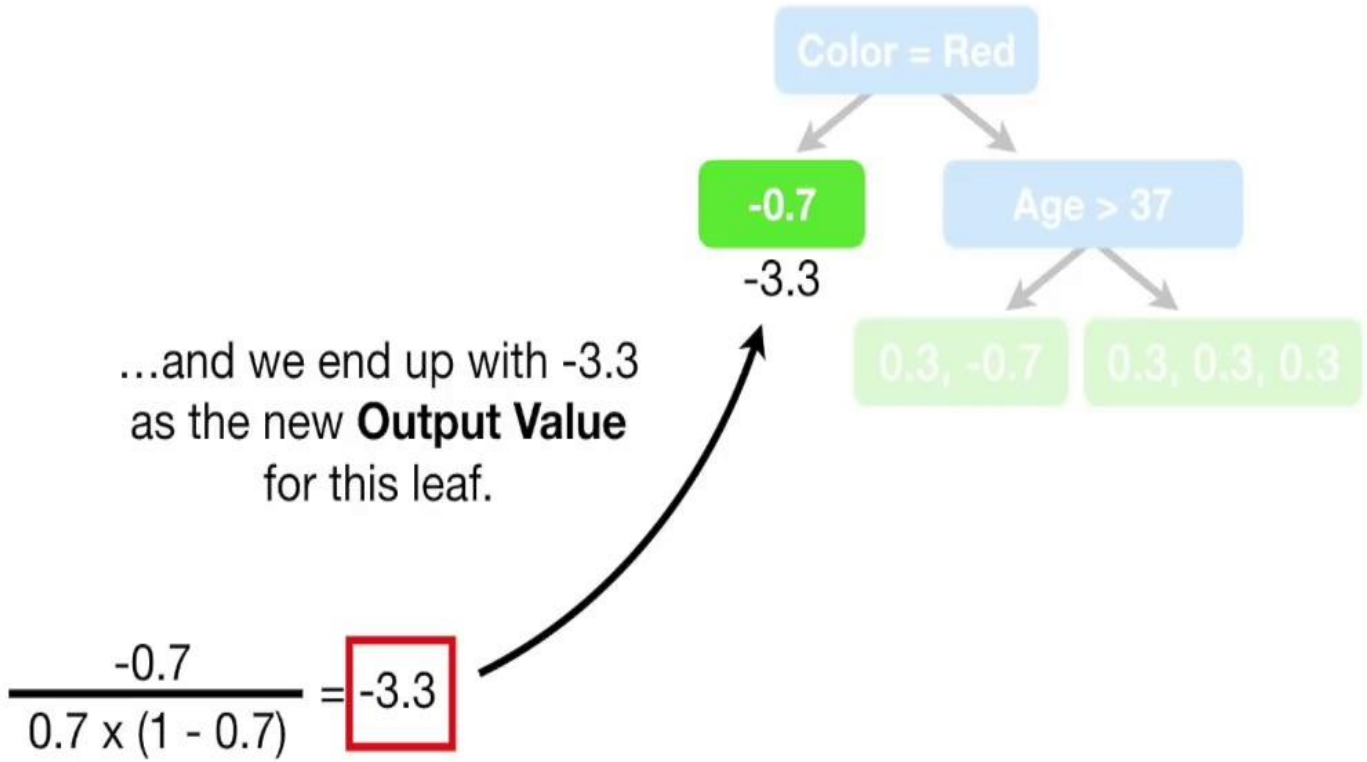
$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$



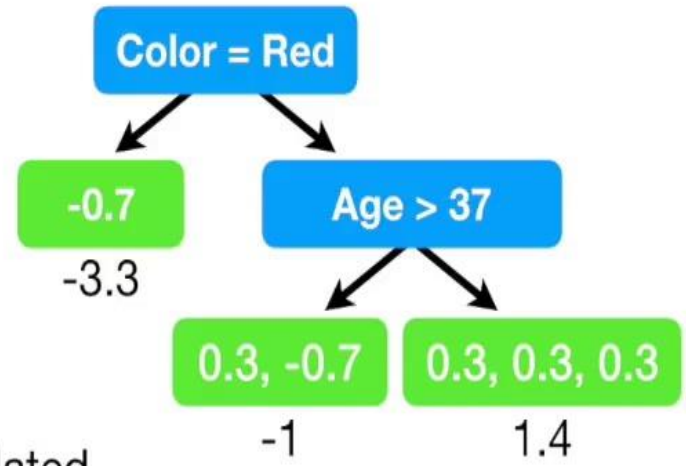
Example


log(4/2) = 0.7
Probability of Loving Troll 2 = 0.7
Color = Red
-0.7
Age > 37
...so we plug that in...
-0.7
0.7 x (1 - 0.7)

Example

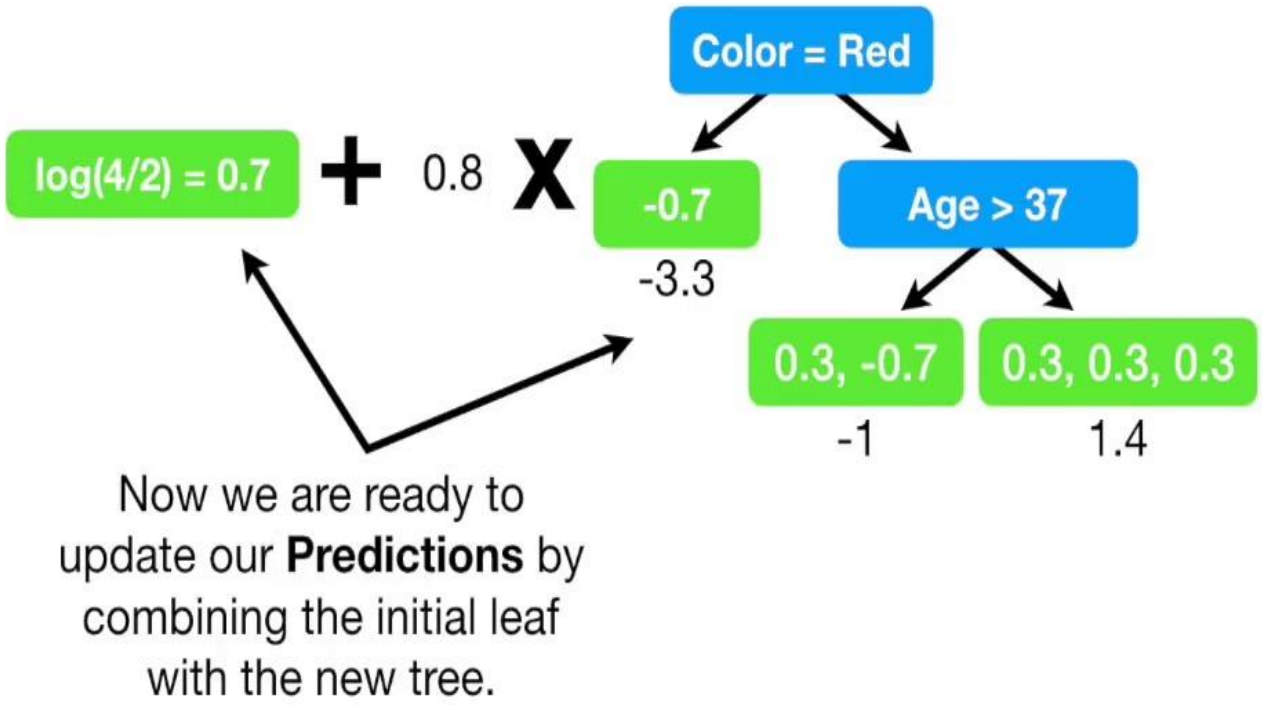

Color = Red
-0.7
-3.3
Age > 37
0.3, -0.7
0.3, 0.3, 0.3
…and we end up with -3.3
as the new **Output Value**
for this leaf.
-0.7
0.7 x (1 - 0.7)
= -3.3

Example



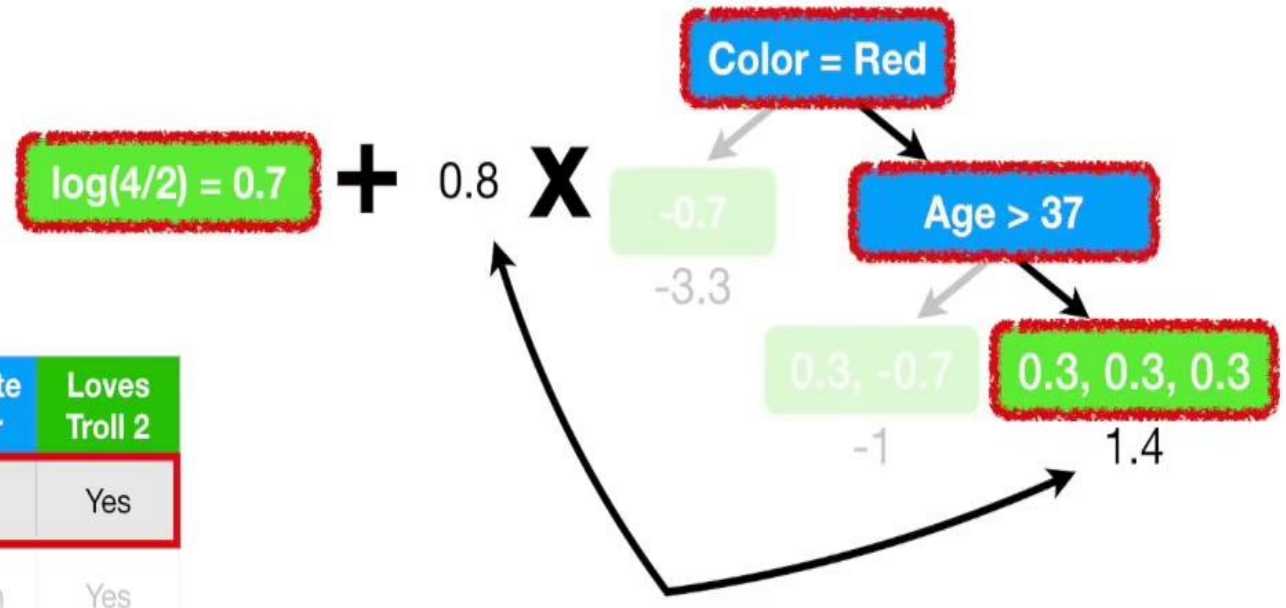
We've calculated **Output Values** for all three leaves in the tree!

Example


log(4/2) = 0.7
+
0.8
X
Color = Red
-0.7
-3.3
Age > 37
0.3, -0.7
-1
0.3, 0.3, 0.3
1.4
Now we are ready to update our **Predictions** by combining the initial leaf with the new tree.

Example

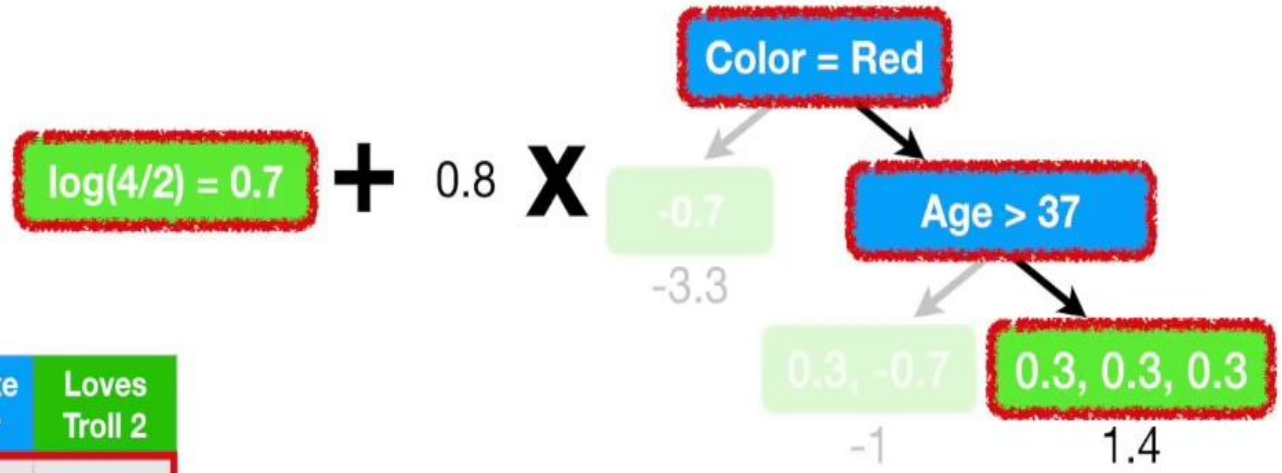
Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes



…plus the **Output Value** from the tree scaled by the **Learning Rate**…

log(odds) Prediction = 0.7 + (0.8 x 1.4)

Example



Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

...and the new **log(odds) Prediction = 1.8**.

log(odds) Prediction = 0.7 + (0.8 x 1.4) = 1.8

Example

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Now we convert the new **log(odds) Prediction** into a **Probability**...

$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

$$\log(\text{odds})\ \text{Prediction} = 0.7 + (0.8 \times 1.4) = 1.8$$

Example

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Now we convert the new **log(odds) Prediction** into a **Probability**...

$$\text{Probability} = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$

$$\log(\text{odds}) \text{ Prediction} = 0.7 + (0.8 \times 1.4) = 1.8$$

Example

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

Now we convert the new **log(odds) Prediction** into a **Probability**...

$$\text{Probability} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$$

$$\text{log(odds) Prediction} = 0.7 + (0.8 \times 1.4) = 1.8$$

Example

log(4/2) = 0.7

Initial Probability of **Loving Troll 2** = 0.7

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
Yes	87	Green	Yes
No	44	Blue	No
Yes	19	Red	No
No	32	Green	Yes
No	14	Blue	Yes

…so we are taking a small step in the right direction since this person **Loves Troll 2**.

$$\text{Probability} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$$

log(odds) Prediction = 0.7 + (0.8 x 1.4) = 1.8

Example

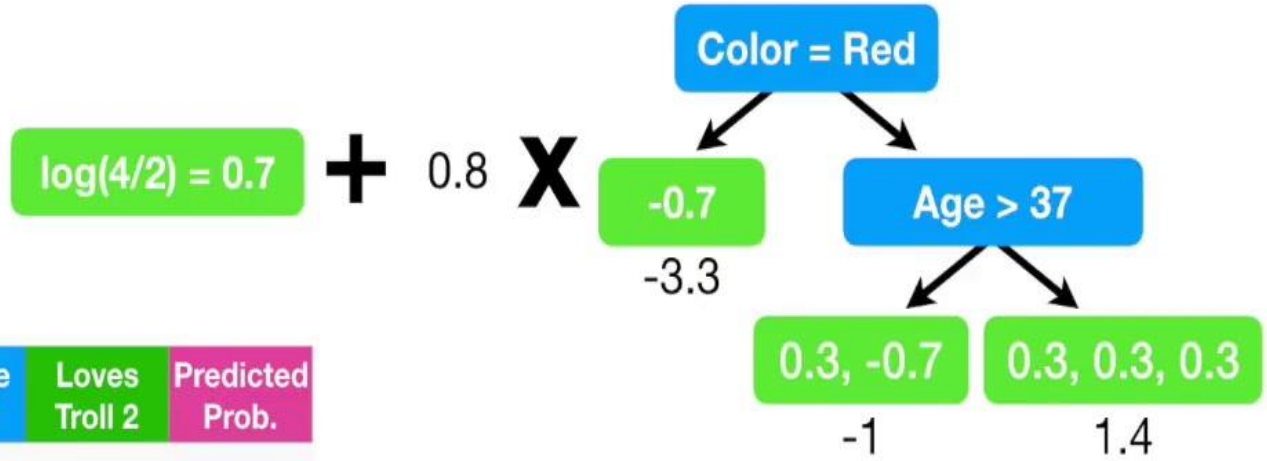
Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.
Yes	12	Blue	Yes	0.9
Yes	87	Green	Yes	
No	44	Blue	No	
Yes	19	Red	No	
No	32	Green	Yes	
No	14	Blue	Yes	

We save the new **Predicted Probability** here.

$$\text{Probability} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$$

log(odds) Prediction = 0.7 + (0.8 x 1.4) = 1.8

Example



Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.
Yes	12	Blue	Yes	0.9
Yes	87	Green	Yes	0.5
No	44	Blue	No	0.5
Yes	19	Red	No	0.1
No	32	Green	Yes	0.9
No	14	Blue	Yes	0.9

Then we calculate the **Predicted Probabilities** for the remaining people.

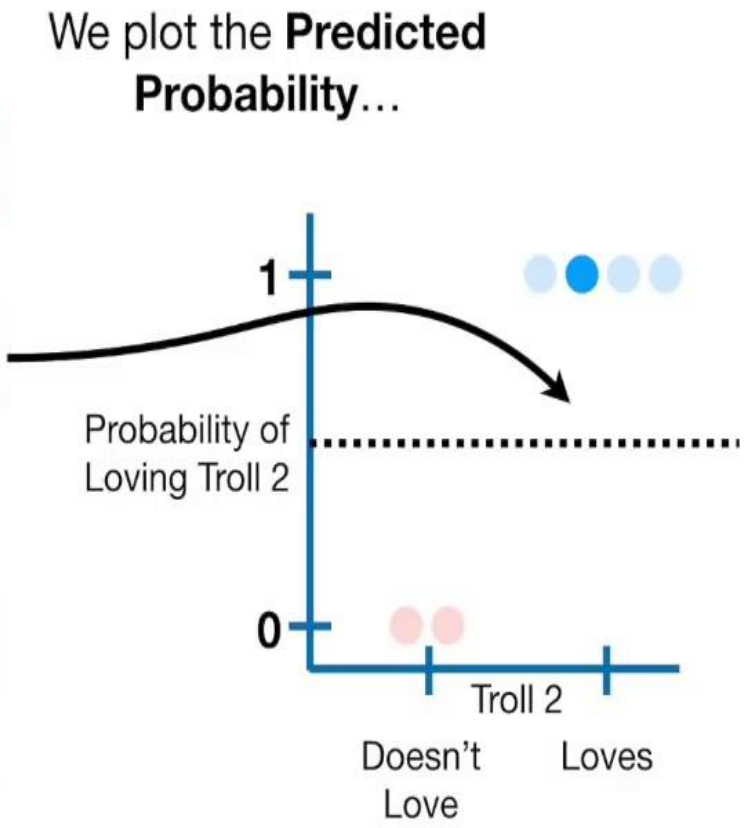
Example

And now, just like before, we calculate the new **Residuals**…

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	
Yes	87	Green	Yes	0.5	
No	44	Blue	No	0.5	
Yes	19	Red	No	0.1	
No	32	Green	Yes	0.9	
No	14	Blue	Yes	0.9	

Example

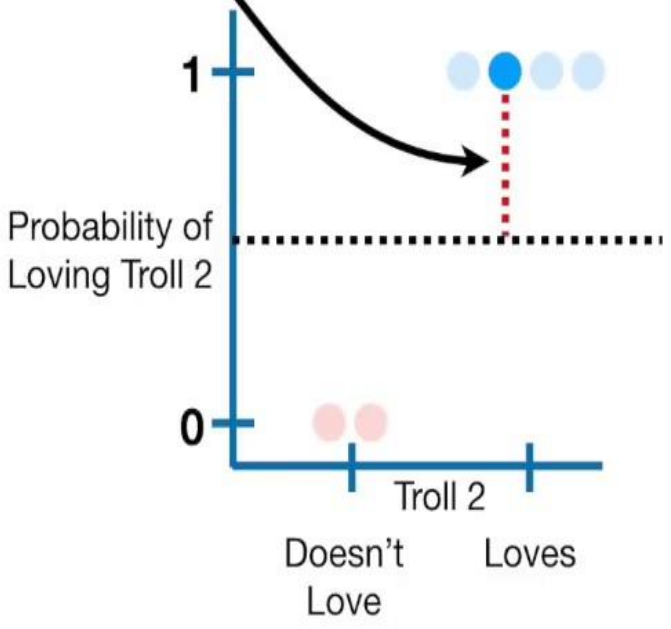
Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	
No	44	Blue	No	0.5	
Yes	19	Red	No	0.1	
No	32	Green	Yes	0.9	
No	14	Blue	Yes	0.9	



Example

...and the **Residual** is the difference. → **Residual = (Observed - Predicted)**

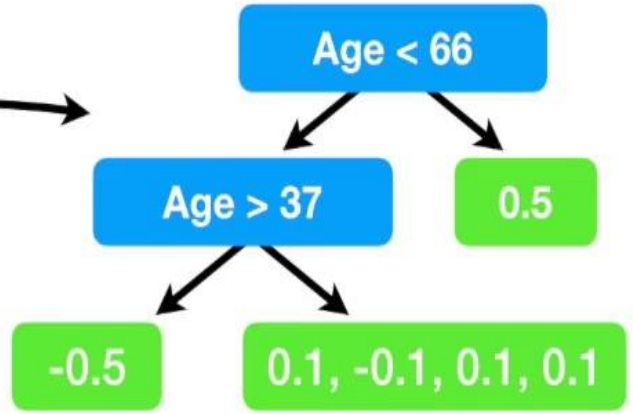
Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	
No	44	Blue	No	0.5	
Yes	19	Red	No	0.1	
No	32	Green	Yes	0.9	
No	14	Blue	Yes	0.9	



Example

Now that we have the **Residuals**, we can build a new tree...

Likes Popcorn	Age	Favorite Color	Loves Troll 2	Predicted Prob.	Residual
Yes	12	Blue	Yes	0.9	0.1
Yes	87	Green	Yes	0.5	0.5
No	44	Blue	No	0.5	-0.5
Yes	19	Red	No	0.1	-0.1
No	32	Green	Yes	0.9	0.1
No	14	Blue	Yes	0.9	0.1



GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,...,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

Step 3: Output $F_M(x)$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$



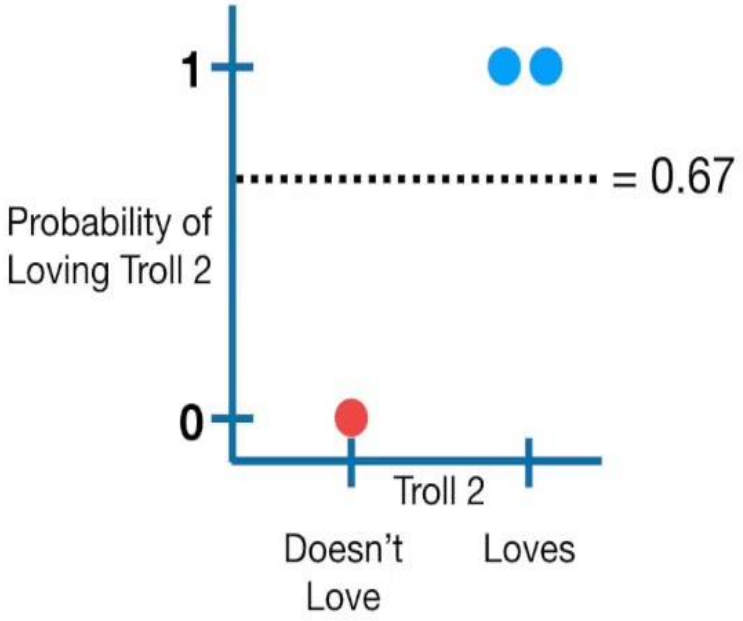
Now we need a differentiable **Loss Function** that will work for **Classification**.

GB Algorithm

Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

Log(Likelihood of the Observed Data given the Prediction) =

$$\sum_{i=1}^{N} y_i \times \log(p) + (1 - y_i) \times \log(1 - p)$$



$$-\left[\textbf{Observed} \times \log(\boldsymbol{p}) + (1 - \textbf{Observed}) \times \log(1 - \boldsymbol{p})\right]$$

$$\sum_{i=1}^{N} \boldsymbol{y_i} \times \log(\boldsymbol{p}) + (1 - \boldsymbol{y_i}) \times \log(1 - \boldsymbol{p})$$

1) -**Observed** x log(***p***) - (1 - **Observed**) x log(1 - ***p***)

2) -**Observed** x log(***p***) - log(1 - ***p***) + **Observed** x log(1 - ***p***)

3) -**Observed** x [log(***p***) - log(1- ***p***)] - log(1 - ***p***)

4) -**Observed** x log(odds) - log(1 - ***p***)

5) -**Observed** x log(odds) + log(1 + $e^{\log(odds)}$)

We converted the **negative log(likelihood)** of the data, which is a function of the predicted probability, ***p***…

…into a function of the predicted **log(odds)**.

This is the Loss function

GB Algorithm

$$\frac{d}{d\ log(odds)} -\textbf{Observed} \times \log(odds) + \log(1 + e^{\log(odds)}) =$$



So let's take the derivative of the **Loss Function** with respect to the predicted **log(odds)**.

GB Algorithm

$$\frac{d}{d\,log(odds)} \text{-Observed} \times \log(odds) + \log(1 + e^{\log(odds)}) =$$

$$= \text{-Observed} + \frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$$

$$= \text{-Observed} + p$$

Indeed, in the previous example we performed

- 1 – 0.7 = 0.3
- 0 – 0.7 = -0.7

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

We have created the initial predicted **log(odds)**, $F_0(x)$…

$$F_0(x) = \log\left(\frac{2}{1}\right)$$

GB Algorithm

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

So this is the initial leaf, ***F₀(x)***.

$$F_0(x) = \log\left(\frac{2}{1}\right)$$

$$= 0.69$$

log(2/1) = 0.69

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

This is just the derivative of the **Loss Function**…

$$\frac{d}{d\ log(odds)} \text{-}\mathbf{Observed} \times \log(odds) + \log(1 + e^{\log(odds)})$$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

This is just the derivative of the **Loss Function**…

$$\frac{d}{d\ log(odds)} \text{-}\mathbf{Observed} \times \log(odds) + \log(1 + e^{\log(odds)})$$

…with respect to the predicted **log(odds)**…

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

$$\frac{d}{d\ log(odds)} \textbf{-Observed} \times \log(odds) + \log(1 + e^{\log(odds)})$$

…and we've already calculated this.

$$= (\textbf{-Observed} + \frac{e^{\log(odds)}}{1 + e^{\log(odds)}})$$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

…and that leaves us with this equation for calculating **Pseudo Residuals**.

$$\left(\text{Observed} - \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}\right)$$

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

NOTE: As we have seen before, we can replace this term with the predicted probability, ***p***…

→ **(Observed - *p*)**

↕

(Observed - $\frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$ **)**

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

…so we can think of the **Pseudo Residuals** as the **Observed** probability minus the **Predicted** probability… → (**Observed** - ***p***)

(**Observed** - $\frac{e^{\log(odds)}}{1 + e^{\log(odds)}}$)

GB Algorithm

Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

Now we are ready for **Part B**, where we will build a regression tree.

GB Algorithm

Now let's do **Part C**.

Step 2: for $m = 1$ to M:

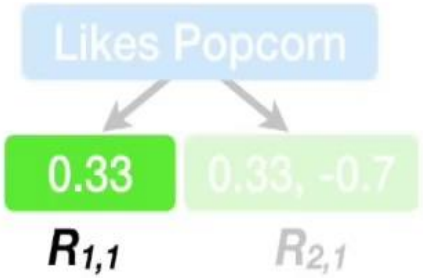
(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

GB Algorithm

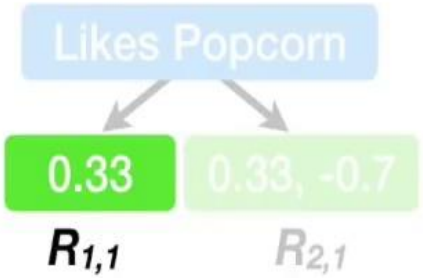
Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No



find the **Output Value** for this leaf.

$$\gamma_{1,1} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

GB Algorithm



Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

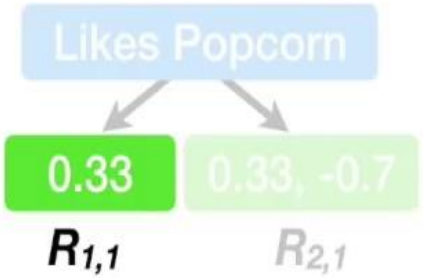
$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$

$$\gamma_{1,1} = \frac{\textbf{Residual}}{\boldsymbol{p} \times (1 - \boldsymbol{p})}$$

0.33/(2/3 x 1/3) = 1.48

GB Algorithm

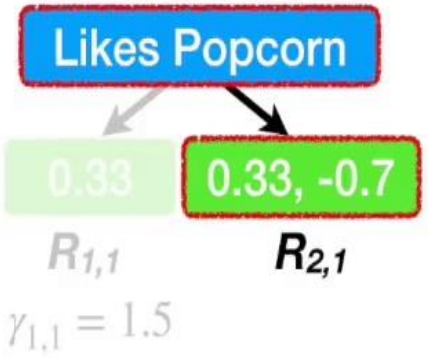
Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No



...and the output value for leaf $R_{1,1}$ is **1.5**.

$\gamma_{1,1} = 1.5$

GB Algorithm

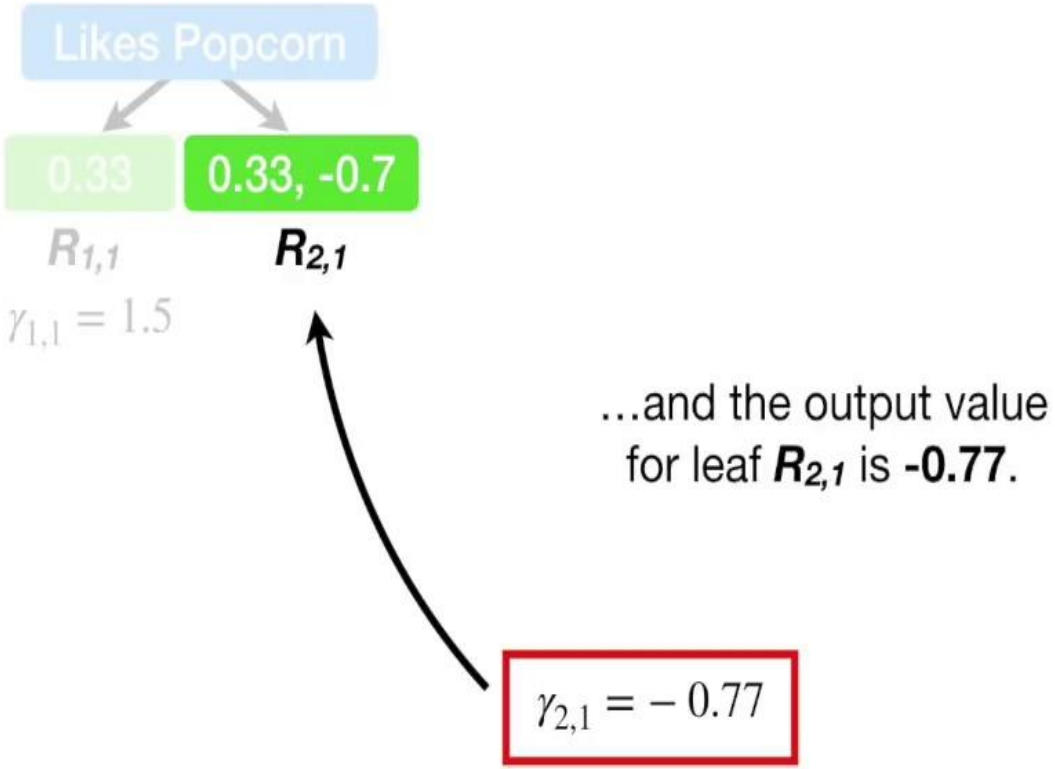


Likes Popcorn	Age	Favorite Color	Loves Troll 2
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

$$\gamma_{2,1} = \frac{\text{Residual}_2 + \text{Residual}_3}{[\,p_2 \times (1 - p_2)\,] + [\,p_3 \times (1 - p_3)\,]}$$

$$\frac{\sum \text{Residual}_i}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right]}$$

GB Algorithm

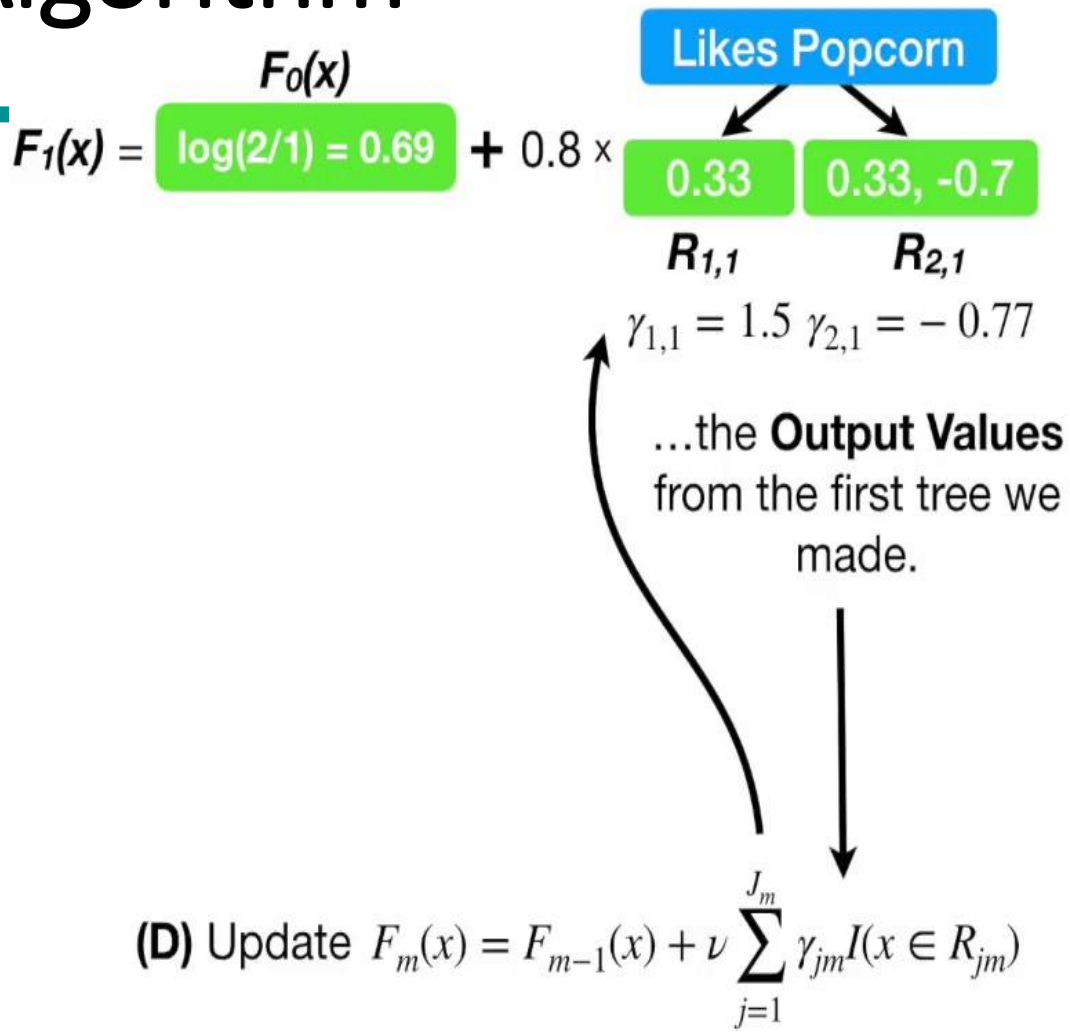


GB Algorithm

In **Part D**, we make a new prediction for each sample.

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

GB Algorithm



Gradient Boost Improved

- XGBoost (Extreme Gradient Boost)
- LightGBM (Light Gradient Boosting Machine)
- CatBoost (Categorical Boost)
- All designed for large complex datasets

Mass	Age	BP	Color	Movie	Car	Hair	etc.
120	23	102	Brown	T2	Ford	Long	...
150	25	98	Brown	Frozen	Kia	Short	...
165	22	130	Black	Spiderman	Ford	Short	...
123	45	98	Red	T2	Kia	Long	...
156	33`	78	Brown	Frozen 2	Ford	Long	...
...	...	...	...	...	...	...	...

XGBoost Characteristics

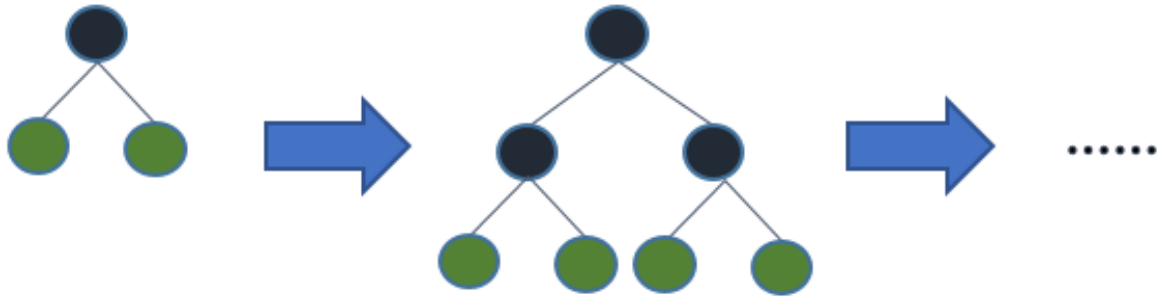
- Gradient Boost
- Regularization
- Approximate Greedy Algorithm
- Weighted Quantile Sketch
- Sparsity-Aware Split Finding
- Parallel Learning
- Cache-Aware Access
- Blocks for Out-of-Core Computation

LightGBM

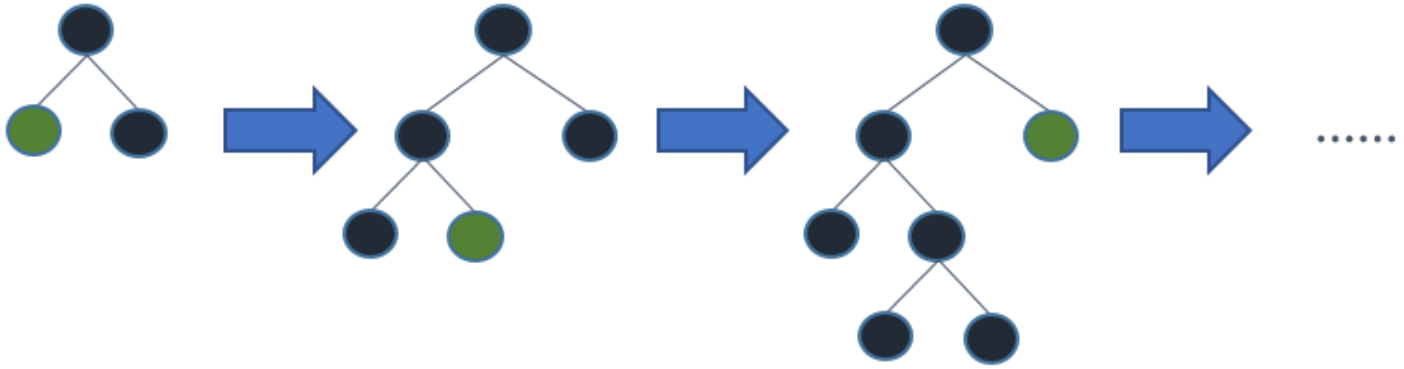
- LightGBM (by Microsoft) is a distributed high-performance framework that uses decision trees for classification, and regression tasks.
- Advantage w.r.t. XGBoost
  - Faster training speed and accuracy resulting from LightGBM being a histogram-based algorithm that performs bucketing of values (also requires lesser memory)
  - Compatible with large and complex datasets but is much faster during training
  - Support for both parallel learning and GPU learning

LightGBM vs XGBoost

- XGBoost: level-wise (horizontal) growth



- LightGBM: out leaf-wise (vertical) growth



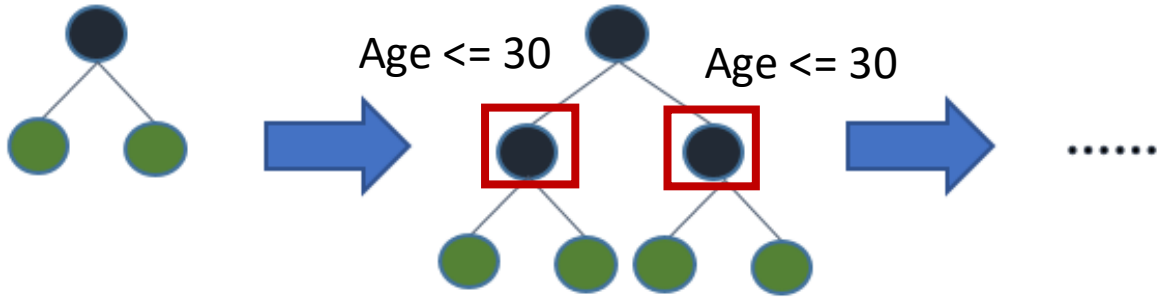
- LightGBM is significantly faster than XGBoost but delivers almost equivalent performance

LightGBM and XGBoost

- Handling Categorical Features
- Handling Missing Values

CatBoost

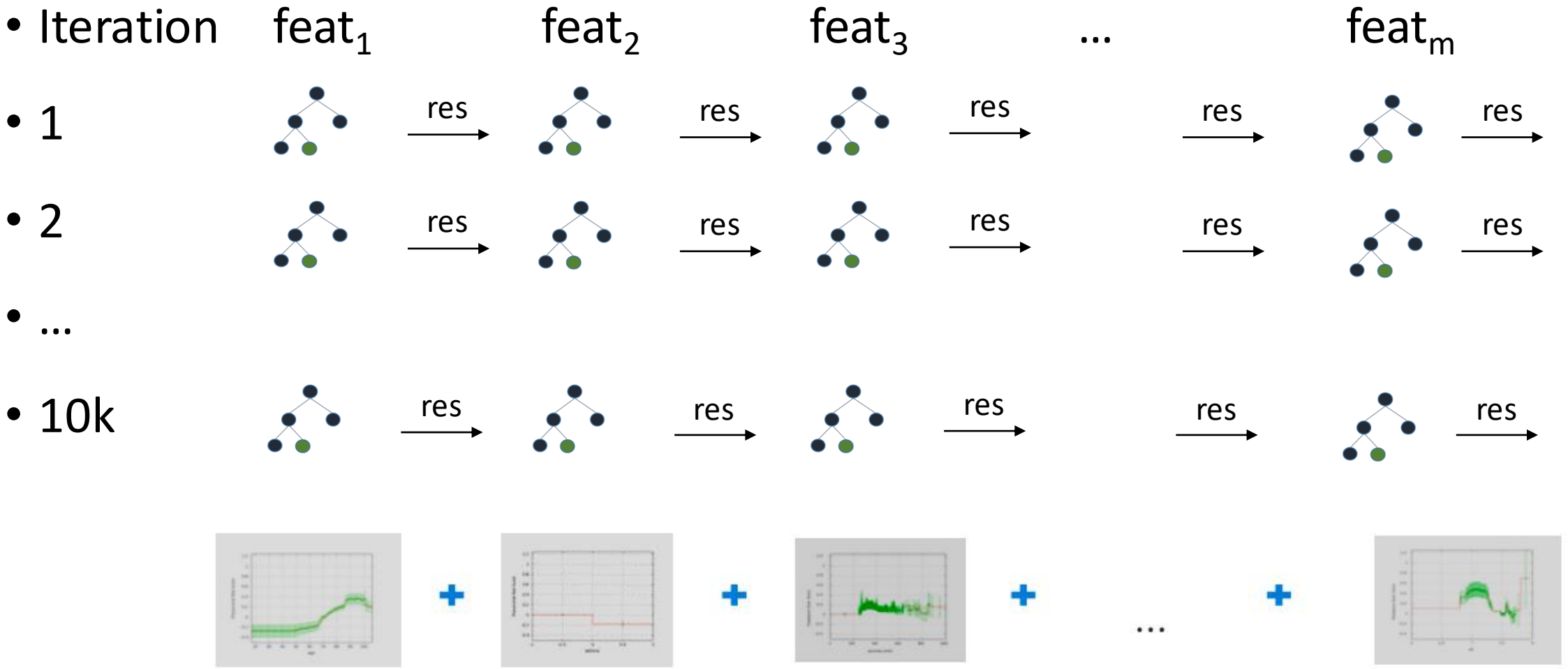
- Categorial Boosting
- Gradient Boosting Machine
- Adopts Ordered Target Encoding procedure for categorical attributes
- Rely on Symmetric Decision Trees, i.e., dedicion trees with the same splitting conditions for nodes at the same lavel



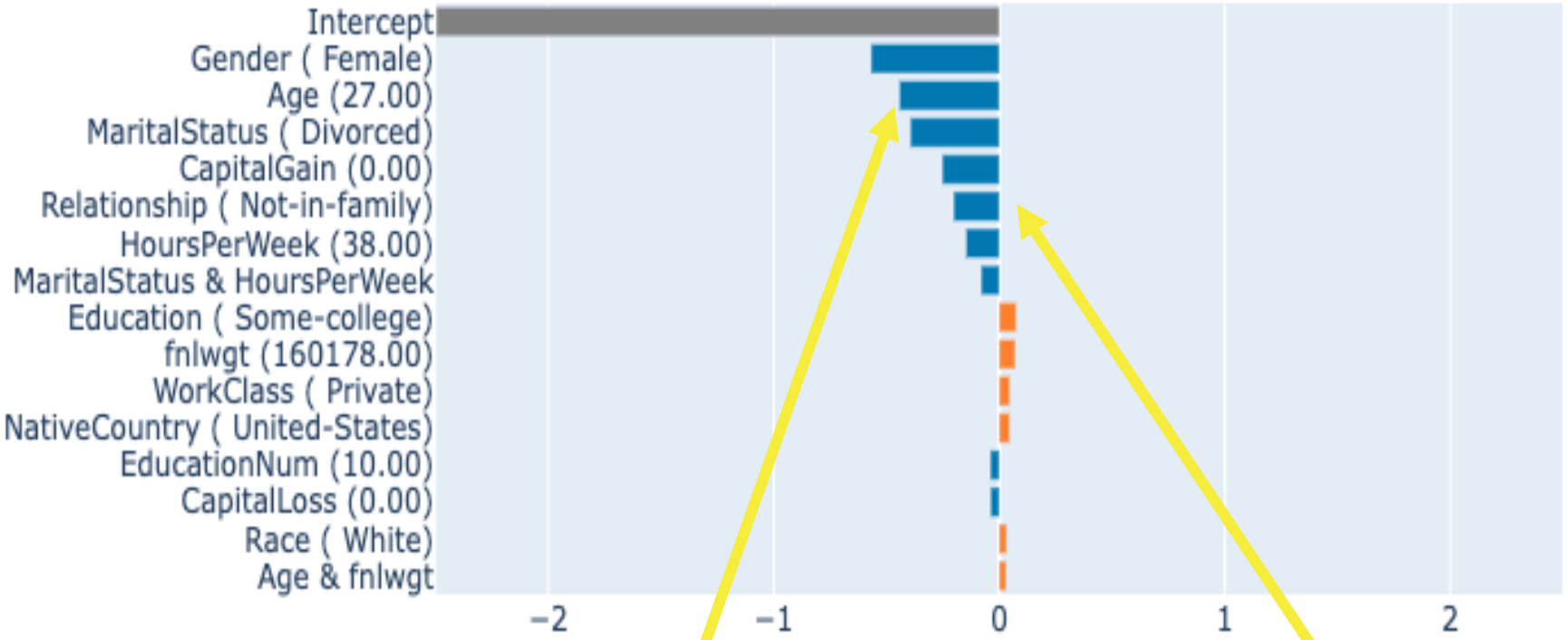
Explainable Boosting Machines (EMB)

- Type of Generalized Additive Models (GAMS)
- Additive Model: $y = f_1(x_1) + f_2(x_2) + \ldots + f_m(x_m)$
- Linear Model: $y = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$
- Complex Model: $y = f(x_1, x_2, \ldots, x_m)$

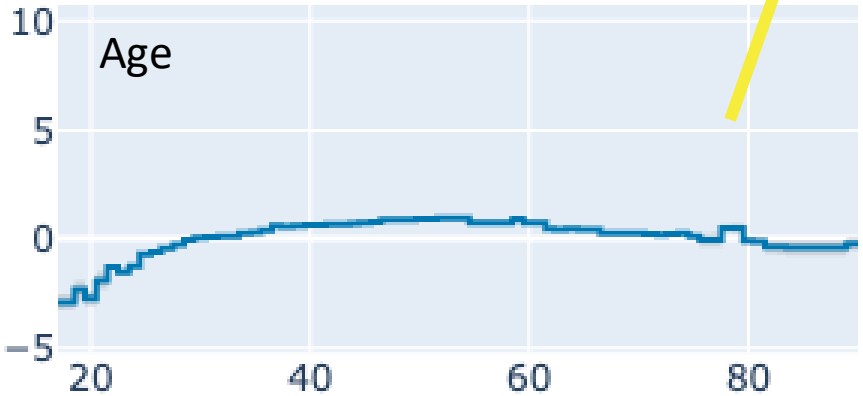
EBM Algorithm Sketch



EBM Prediction Time

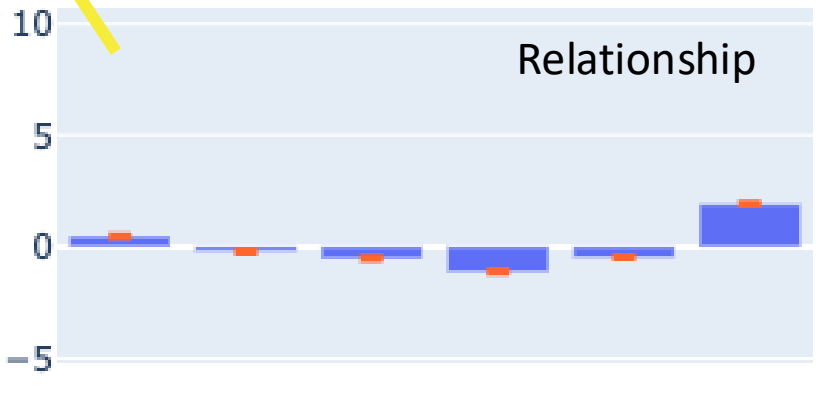


… +



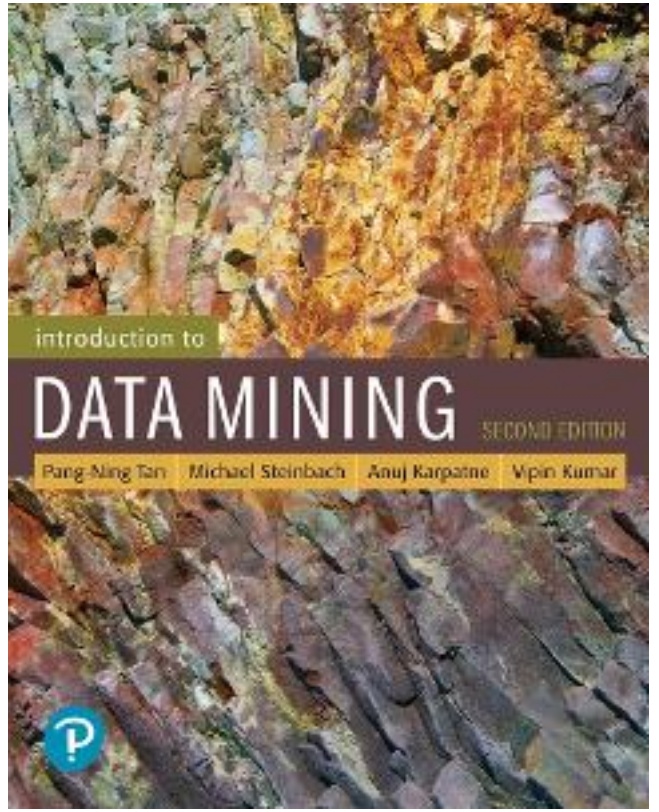
Contribution to Prediction

+ … + … +



References

- Ensemble Methods. Chapter 5.6. Introduction to Data Mining.
- Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent. Advances in neural information processing systems, 12.
- Friedman, J. H. (2002). Stochastic gradient boosting. Computational statistics & data analysis, 38(4), 367-378.
- Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.



Exercises - Ensemble

Exercise Ensemble

- We have 3 independent models for the same data, with poor performances
  - Error1 = 45%
  - Error2 = 40%
  - Error3 = 35%
- Is it better to use Model3 alone or to make bagging with all the three models?

Exercise Ensemble - Solution

- TODO: compute the probability of error of the ensemble
- Standard formula (case for 25 models):

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i}$$

- Implicitly enumerates all cases with more errors than correct answers (i >= 13 errors against 25-i <=12 correct ones)
  - However, it works only when all models have the same error ε
- Here we have to explicitly enumerate all cases

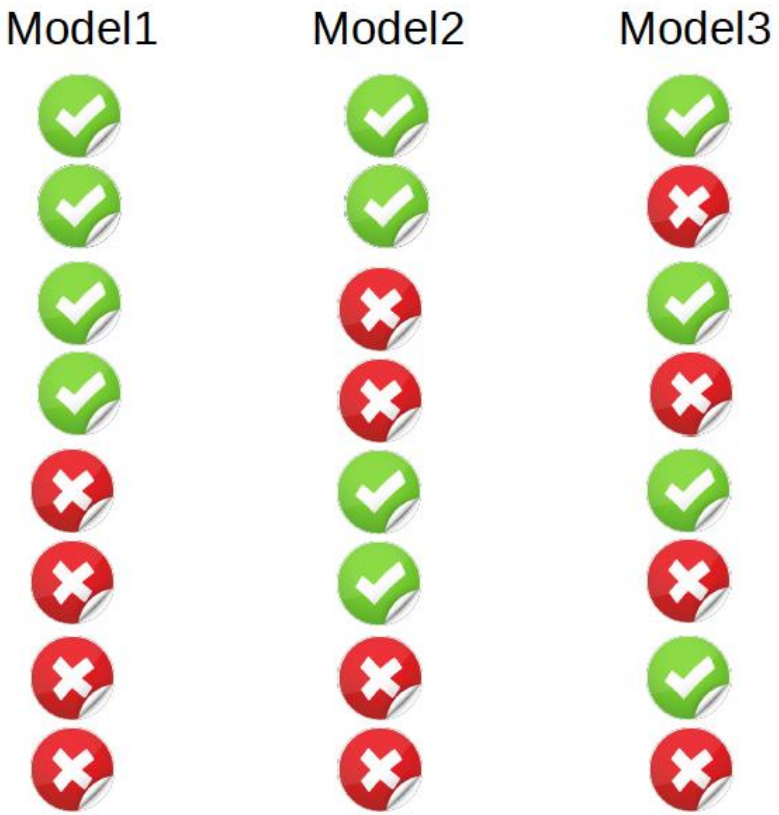
Exercise Ensemble - Solution

- Probability of success ✅ and failure ❌ of each:

	ε	1-ε
– Model1 =	❌ .45	✅ .55
– Model2 =	❌ .40	✅ .60
– Model3 =	❌ .35	✅ .65

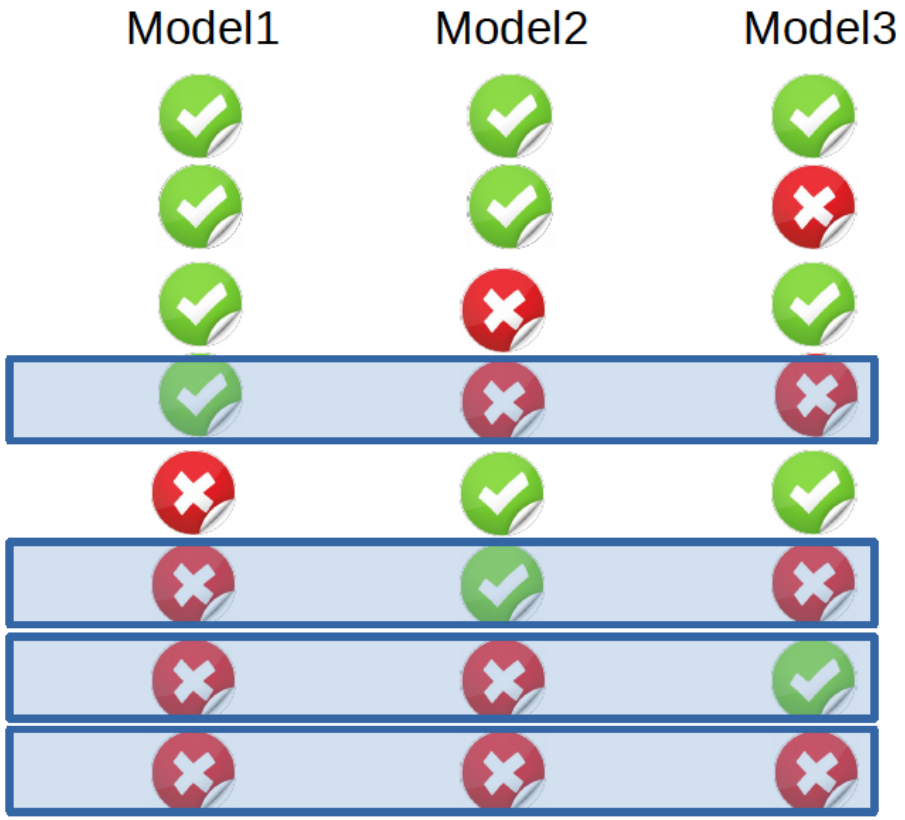
Exercise Ensemble - Solution

- We have 8 possible cases



Exercise Ensemble - Solution

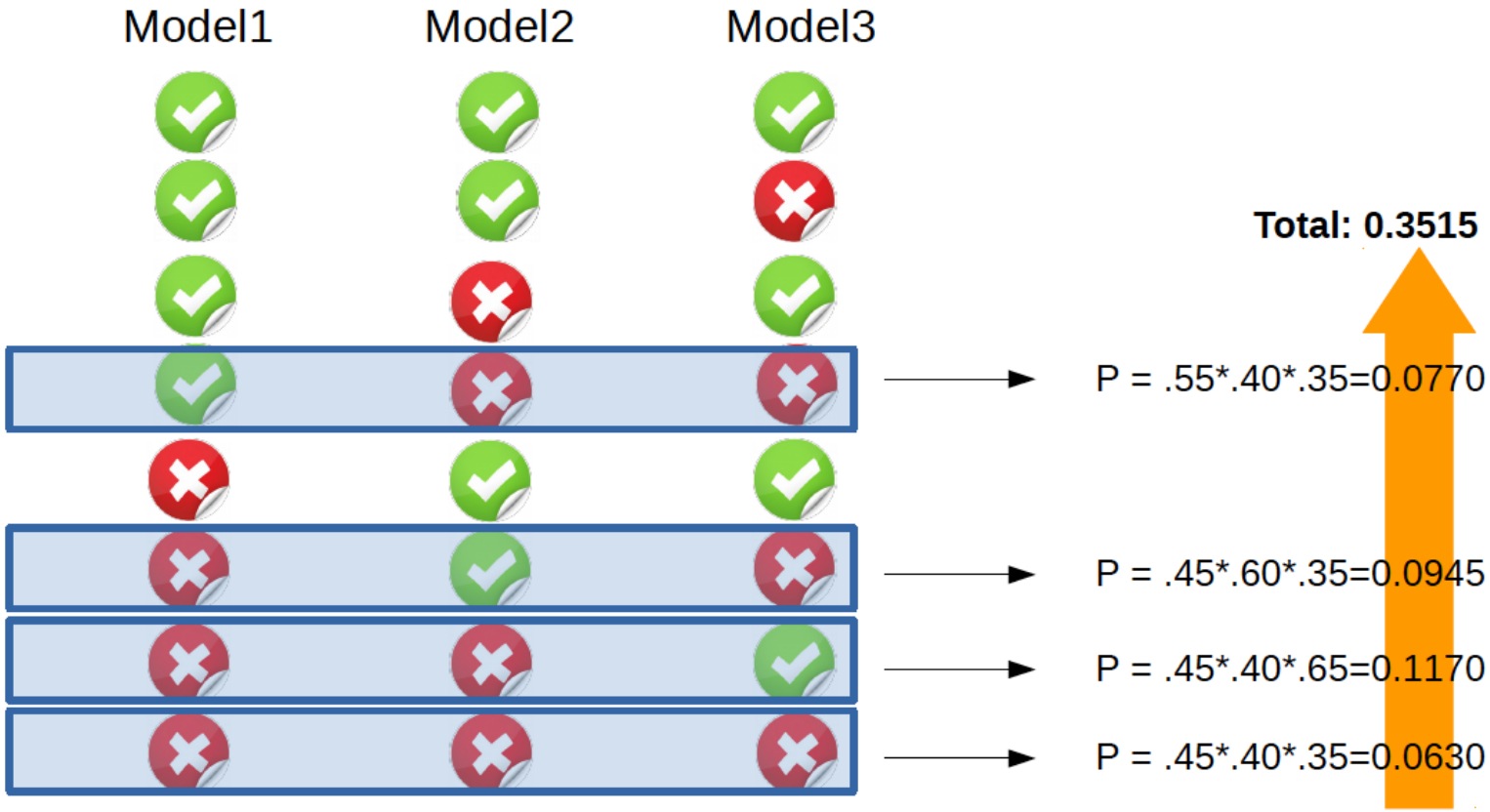
- We have 8 possible cases



Exercise Ensemble - Solution

- Probability of success ✓ and failure ✗ of each:
  - Model1 = ✗ .45 ✓ .55
  - Model2 = ✗ .40 ✓ .60
  - Model3 = ✗ .35 ✓ .65

- We have 8 possible cases



Exercise Ensemble - Solution

- Outcome:
  - The “expert” model (Model3) has 35% of error
  - The bagging model has 35.15% of error
  - In this specific case Bagging is not better than the “expert” alone...

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Gain Function = Misclassification Error

1. Find the best split for AdaBoost
2. Which is the value of the importance (alpha) for the best split?
3. Which are the values of the new weights?
4. Which are the values of the new weights normalized?
5. Provide a correct training for the next iteration based on the new weights.

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

State	Y	N
Italy	2/3	1/3
France	1/2	1/2
Germ	1/2	1/2

Gain Function = Misclassification Error

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

State	Y	N
Italy	2/3	1/3
France	1/2	1/2
Germ	1/2	1/2

Child Imp = 3/7*1/3 + 2/7*1/2 + 2/7*1/2 = 3/7

Gain Function = Misclassification Error

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Gain Function = Misclassification Error

State	Y	N
Italy	2/3	1/3
France	1/2	1/2
Germ	1/2	1/2

Child Imp = 3/7*1/3 + 2/7*1/2 + 2/7*1/2 = 3/7

Travel	Y	N
Yes	3/4	1/4
No	1/3	2/3

Child Imp = 4/7*1/4 + 3/7*1/3 = 2/7

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Gain Function = Misclassification Error

State	Y	N
Italy	2/3	1/3
France	1/2	1/2
Germ	1/2	1/2

Child Imp = 3/7*1/3 + 2/7*1/2 + 2/7*1/2 = 3/7

Travel	Y	N
Yes	3/4	1/4
No	1/3	2/3

Child Imp = 4/7*1/4 + 3/7*1/3 = 2/7

Sex	Y	N
M	2/4	2/4
F	2/3	1/3

Child Imp = 4/7*2/4 + 3/7*1/3 = 3/7

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Gain Function = Misclassification Error

State	Y	N
Italy	2/3	1/3
France	1/2	1/2
Germ	1/2	1/2

Child Imp = 3/7*1/3 + 2/7*1/2 + 2/7*1/2 = 3/7

Travel	Y	N
Yes	3/4	1/4
No	1/3	2/3

Child Imp = 4/7*1/4 + 3/7*1/3 = 2/7

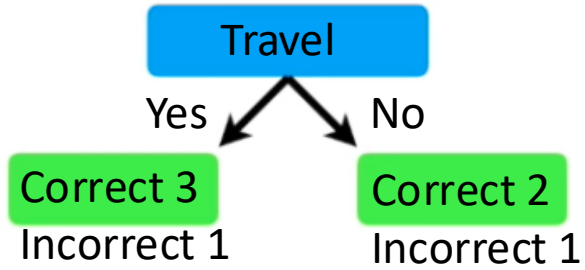
Sex	Y	N
M	2/4	2/4
F	2/3	1/3

Child Imp = 4/7*2/4 + 3/7*1/3 = 3/7

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Split by Travel



Error = 2/7

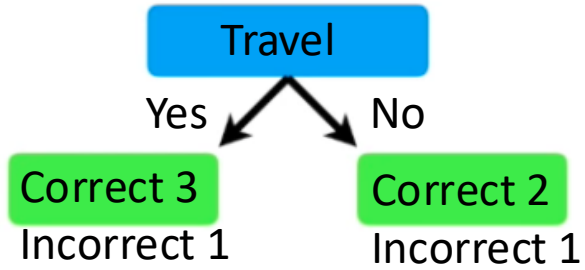
Gain Function = Misclassification Error

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7		
Germ	Yes	M	Y	1/7		
Italy	No	M	N	1/7		
Italy	No	F	Y	1/7		
France	Yes	M	Y	1/7		
Germ	No	F	N	1/7		
France	Yes	M	N	1/7		

Gain Function = Misclassification Error

Split by Travel



Error = 2/7

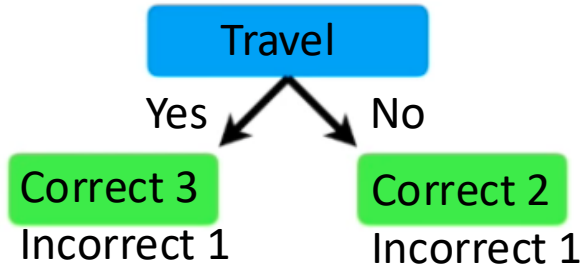
Alpha = 1/2 * ln((1-2/7)/(2/7)) = 1/2 ln(5/2) = 0.458

Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7	0.63	
Germ	Yes	M	Y	1/7	0.63	
Italy	No	M	N	1/7	0.63	
Italy	No	F	Y	1/7	1.58	
France	Yes	M	Y	1/7	0.63	
Germ	No	F	N	1/7	0.63	
France	Yes	M	N	1/7	1.58	

Gain Function = Misclassification Error

Split by Travel



Error = 2/7

Alpha = 1/2 * ln((1-2/7)/(2/7)) = 1/2 ln(5/2) = 0.458

$1/7 * e^{0.458} = 1.58$ misclassified

$1/7 * e^{-0.458} = 0.63$ correctly classified

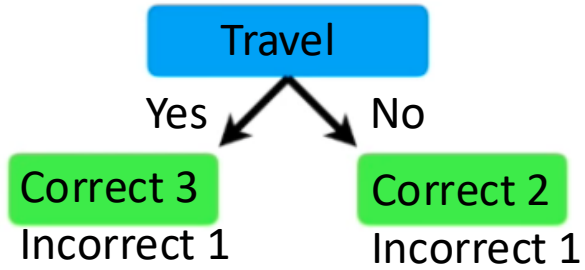
Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7	0.09	
Germ	Yes	M	Y	1/7	0.09	
Italy	No	M	N	1/7	0.09	
Italy	No	F	Y	1/7	0.23	
France	Yes	M	Y	1/7	0.09	
Germ	No	F	N	1/7	0.09	
France	Yes	M	N	1/7	0.23	

Z = 0.91

Gain Function = Misclassification Error

Split by Travel



Error = 2/7

Alpha = 1/2 * ln((1-2/7)/(2/7)) = 1/2 ln(5/2) = 0.458

$1/7 * e^{0.458} = 0.23$ misclassified

$1/7 * e^{-0.458} = 0.09$ correctly classified

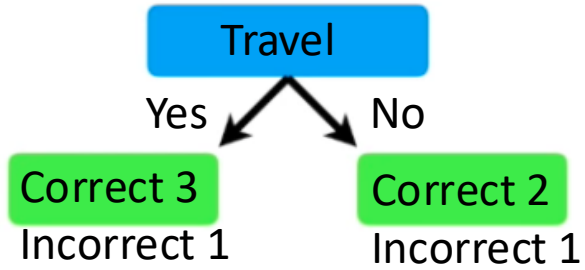
Exercise AdaBoost

state	travel	sex	class	weight	new weight	norm weight
Italy	Yes	F	Y	1/7	0.09	0.09
Germ	Yes	M	Y	1/7	0.09	0.09
Italy	No	M	N	1/7	0.09	0.09
Italy	No	F	Y	1/7	0.23	0.25
France	Yes	M	Y	1/7	0.09	0.09
Germ	No	F	N	1/7	0.09	0.09
France	Yes	M	N	1/7	0.23	0.25

Z = 0.91

Gain Function = Misclassification Error

Split by Travel



Error = 2/7

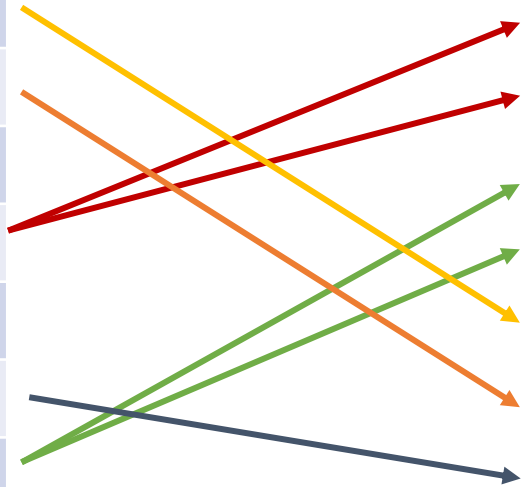
Alpha = 1/2 * ln((1-2/7)/(2/7)) = 1/2 ln(5/2) = 0.458

$1/7 * e^{0.458} = 0.23$ misclassified

$1/7 * e^{-0.458} = 0.09$ correctly classified

Exercise AdaBoost

state	travel	sex	class	weight
Italy	Yes	F	Y	0.09
Germ	Yes	M	Y	0.09
Italy	No	M	N	0.09
Italy	No	F	Y	0.25
France	Yes	M	Y	0.09
Germ	No	F	N	0.09
France	Yes	M	N	0.25



state	travel	sex	class	weight
Italy	No	F	Y	1/7
Italy	No	F	Y	1/7
France	Yes	M	N	1/7
France	Yes	M	N	1/7
Italy	Yes	F	Y	1/7
Germ	Yes	M	Y	1/7
Germ	No	F	N	1/7

Exercise AdaBoost - Classify

state	travel	sex	class
Italy	Yes	F	
Germ	Yes	M	
France	No	M	

If travel = Yes then Y else N - alpha = 0.46
If sex = F then Y else N - alpha = 0.53
If state = Germ then Y else N - alpha = 0.32

Exercise AdaBoost - Classify

state	travel	sex	class
Italy	Yes	F	Y
Germ	Yes	M	Y
France	No	M	N

1. N 0.32 Y 0.46 Y 0.53 -> Y
2. N 0.53 Y 0.32 + Y 0.46 -> Y
3. N 0.32 + N 0.46 + N 0.53 -> N

If travel = Yes then Y else N - alpha = 0.46
If sex = F then Y else N - alpha = 0.53
If state = Germ then Y else N - alpha = 0.32