AI Course

Dr. Mürsel Taşgın

Boosting

Boosting

Boosting a general learning paradigm where *weak learners* put together to a *strong learner*

The original **Boosting Algorithm** was proposed as an answer to a theoretical question in PAC learning. *[The Strength of Weak Learnability; Schapire, 89]*

Consequently, Boosting has interesting theoretical implications, e.g., on the relations between PAC learnability and compression.

- If a concept class is efficiently PAC learnable then it is efficiently PAC learnable by an algorithm whose required memory is bounded by a polynomial in N, *size* c and $\log(1/\varepsilon)$.
- There is no concept class for which efficient PAC learnability requires that the entire sample be contained in memory at one time – there is always another algorithm that "forgets" most of the sample.

Boosting – *PAC Learning*

The key contribution of Boosting has been practical, as a way to compose a *good learner* from many ***weak learners.***

It is a member of a family of **Ensemble Algorithms**, but has stronger guarantees than others.

Check-out: *Statistical learning theory*

Aim: Take enough samples so that probability of a positive example being predicted errorneously as negative is at most ϵ

→ Simple concepts don't divide data to many sets, *few datasets*

→ Failing on many concepts is very very low

→ Few ways to fail, then there is a bound on way to fail → Fail to fail!

Boosting – *PAC Learning*

- How many training examples *N* should we have, such that with probability at least $1-\delta$, hypothesis *h* has error at most ϵ?

- Set of instances *X* → *data points, observations*
- Set of hypothesis *H* → *our hypothesis about data*
- Set of concepts *C* → *underlying concept that generates data*

- *The goal is to achieve low generalization error with high probability*

$$\Pr(Error(h) \le \epsilon) > 1-\delta$$

$$\Pr(Error_{true} > \epsilon) < |H|e^{-\epsilon m}$$

Boosting – *PAC Learning*

- Suppose we have a classifier that has *training error* 0 (zero) and *test error* (true error) greater than ϵ *(misclassification error)*

$$\boxed{\Pr(Error_{true} > \epsilon)}$$

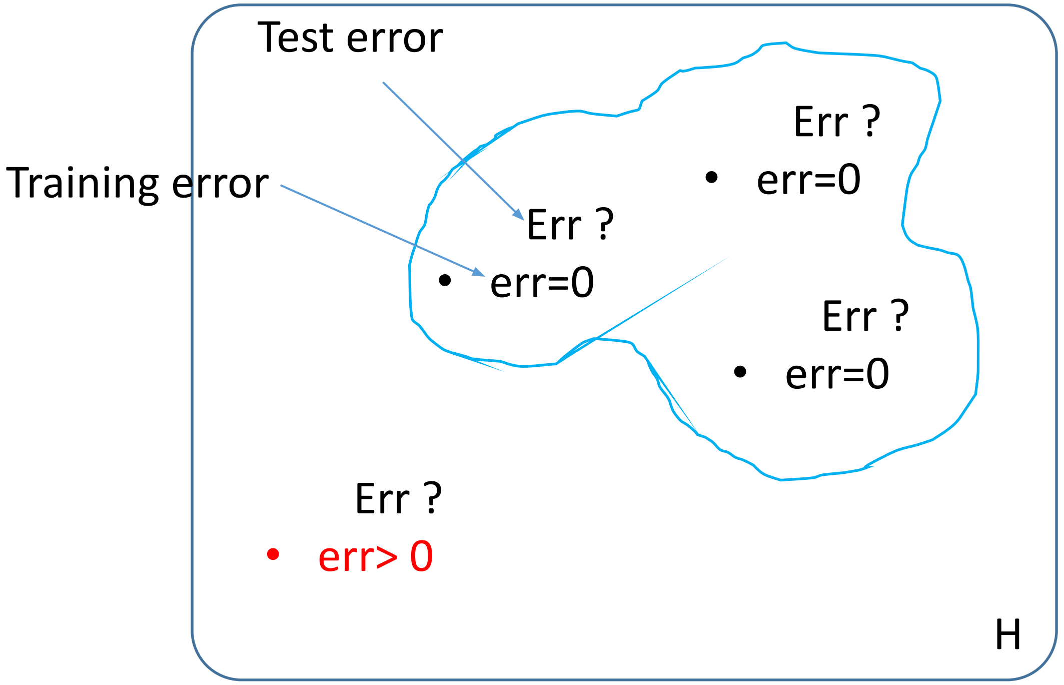
- What is the chance that this classifier makes a single correct classification?

$$\boxed{1 - \epsilon}$$

- What about all of the ***m*** points?

$$\boxed{(1-\epsilon)^m < |\mathrm{H}|e^{-\epsilon m}}$$

Hypothesis space



What is the bound of error, given that classifier classifies all training data correctly?

$$\boxed{\Pr(Error_{true} > \epsilon) < |H|e^{-\epsilon m}}$$

Boosting – *PAC Learning*

Sample complexity

How many data points do I need to guarantee approximately correct classifier?

If we want this upper bound (*probability*) to be at most δ

$$|\mathrm{H}|\mathrm{e}^{-\epsilon m} \leq \delta$$

then

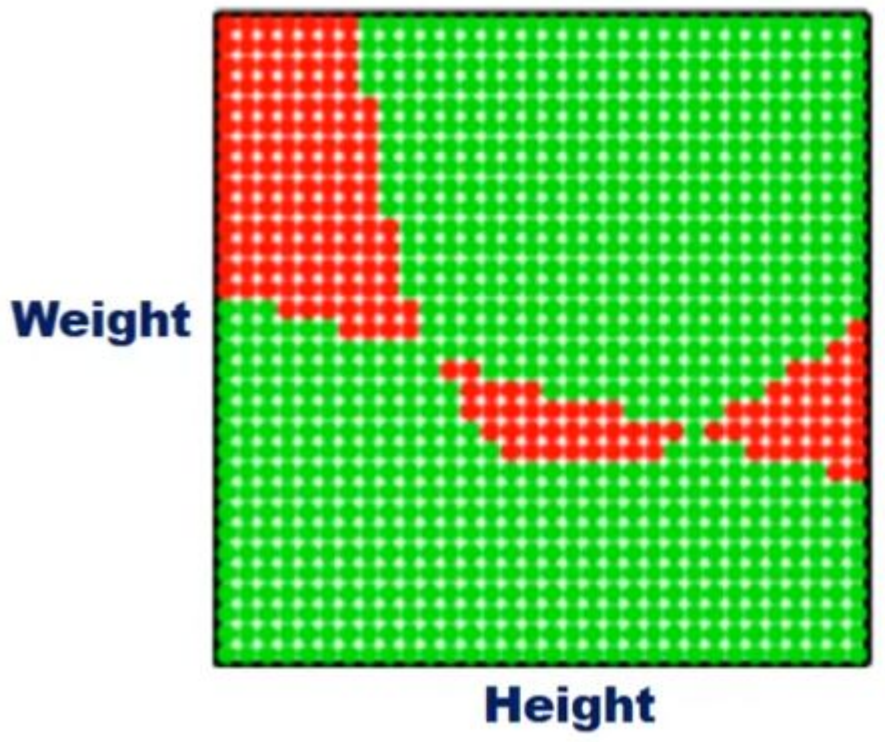
$$\log|H| - \epsilon m \leq \mathrm{l}$$

$$m \geq \frac{1}{\epsilon}\ (\ln|H| + \ln\left(\frac{1}{\delta}\right)$$

$$\downarrow$$

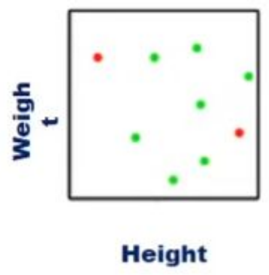
Number of training examples

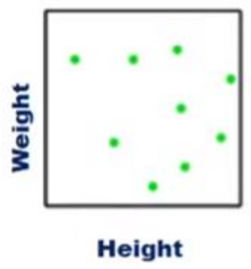
Boosting – *PAC Learning*



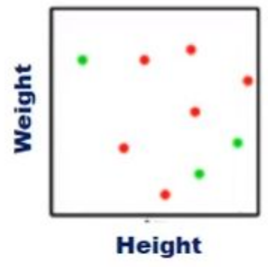
Boosting – *PAC Learning*

Samples from data





⋮



Error of different hypothesis

Error(H1)	Error(h2)
0.04	0.04
0.03	0.035
0.09	0.039
0.06	0.06
0.025	0.025
0.049	0.059
0.04	0.04
0.03	0.03
0.05	0.55
0.043	0.043

$\varepsilon = 0.05$ $\delta = 0.20$

P (H1) = 8/10=0.80

P (H1) = 8/10=0.80 >=1- 0.20

Hence H1 is probably approximately correct

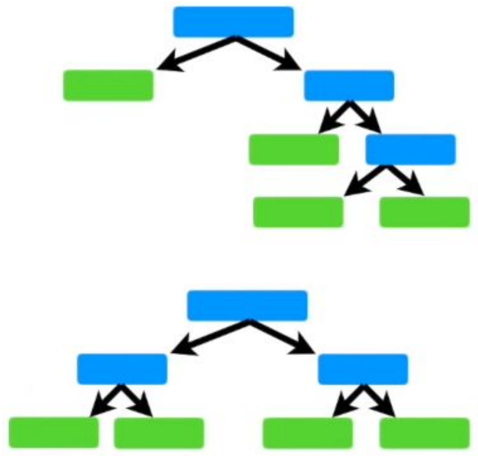
P(H2)= 7/10=0.70

P(H2)= 7/10=0.70 <1-0.20

Hence H2 is not probably approximately correct

Boosting – *AdaBoost*

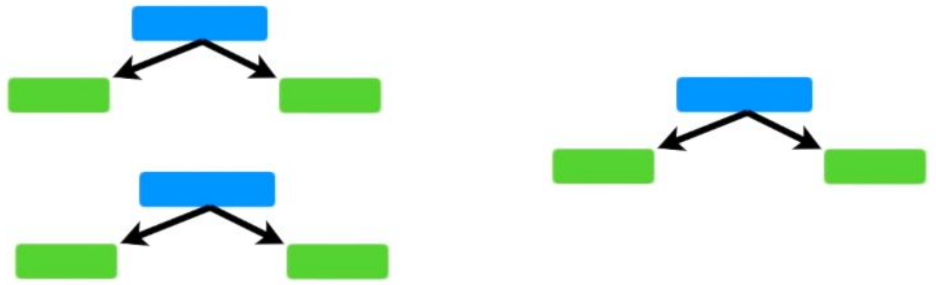
- In a Random Forest, each time we build a complete tree.



Full size decision tree decides using the full tree

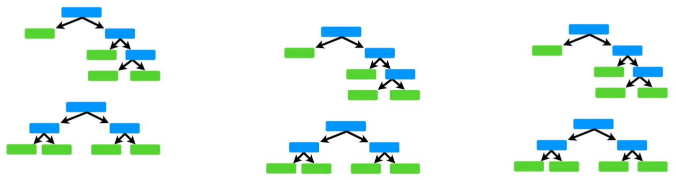
- In AdaBoost, we build forest of stump *weak learner*

A stump can only use one variable to make a decision



Stump: One node with two leaves

Boosting – *AdaBoost*



Random Forest

- Uses full trees to make individual decisions
- Each tree has equal vote on final decision
- Each tree is built independent of each other
- The order of tree creation is not important

AdaBoost

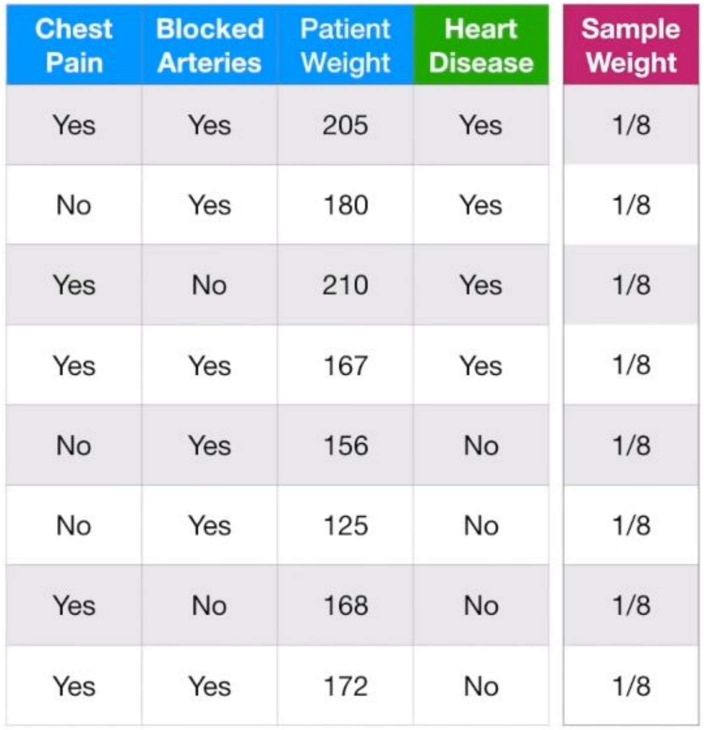
- Uses stumps to make individual decisions
- Some stumps has more saying in voting (not equal)
- Order of stump creation is important
- Errors that the first stump makes, influence the second stump is made
- Combines a lot weak learners

Boosting – *AdaBoost*

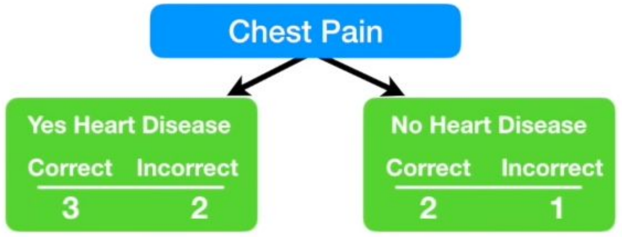
Building stumps in AdaBoost

- Sample from data using ***sample weights***
- Calculate *total errors* of each stump, calculate the *amount of say for* each stump
- Use the first stump with lowest total error as classifier
- Update ***sample weights*** for next round
  - Incorrectly classified samples will get higher weights, correctly classified will have lower weights
- Remove the used feature column (and its stump) from samples
- Iterate

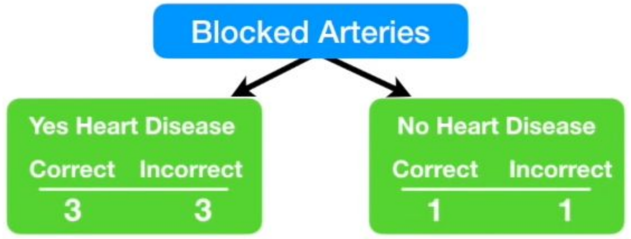
Boosting – *AdaBoost*



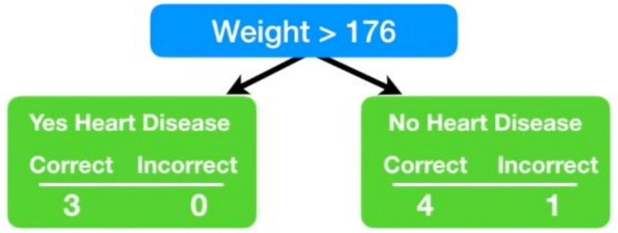
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



Gini Index = 0.47

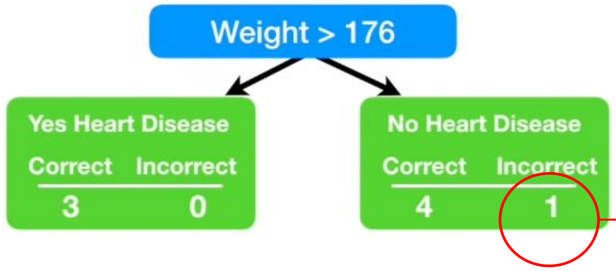


Gini Index = 0.5

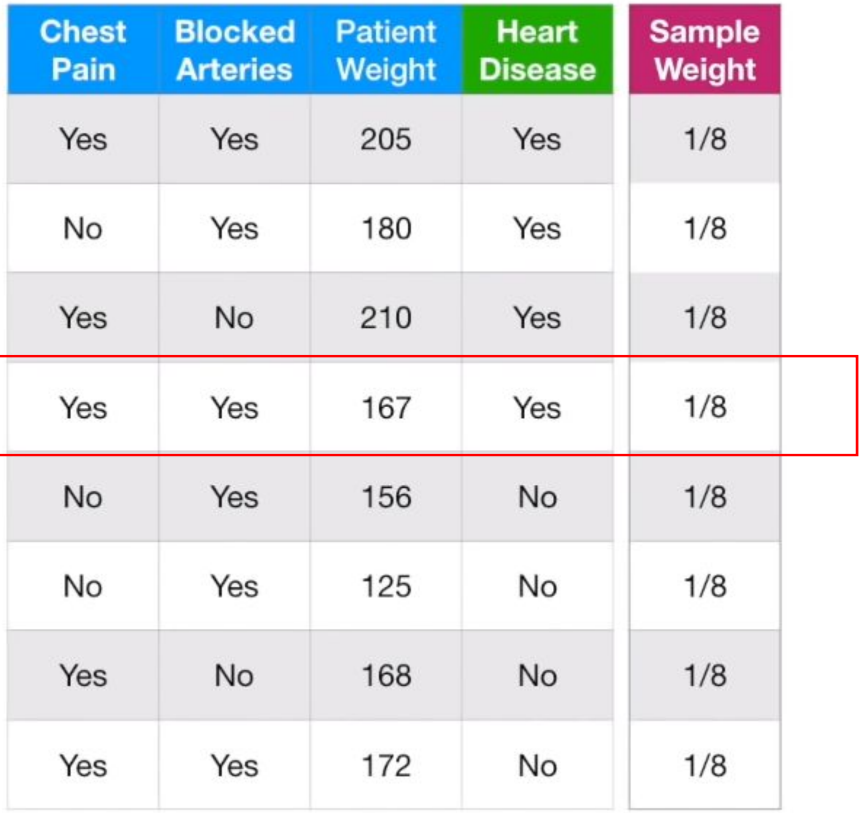


Gini Index = 0.2 $\longrightarrow$ Lowest, select this as the first stump!

Boosting – *AdaBoost*



Emphasize the weight of this incorrect classification so that new stump will make a better job to classify this correct

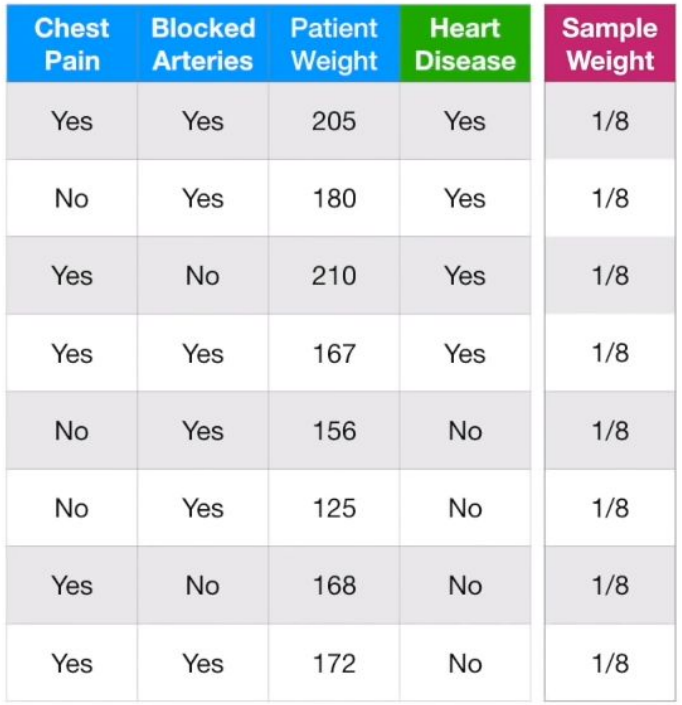


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

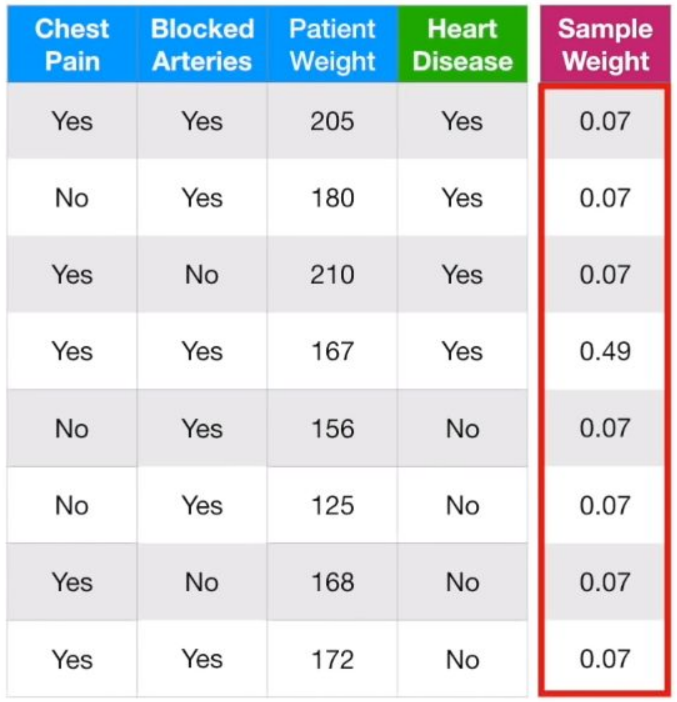
- Increase sample weights of incorrectly classified samples
- Decrease sample weights of correctly classified samples

Incorrectly classified data points will be sampled more will increase the chance to be classified correctly by next stump

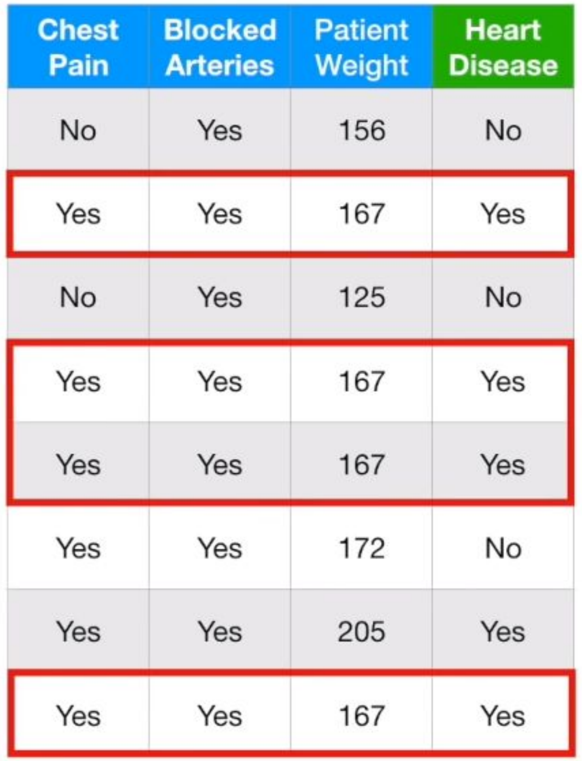
Boosting – *AdaBoost*



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



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

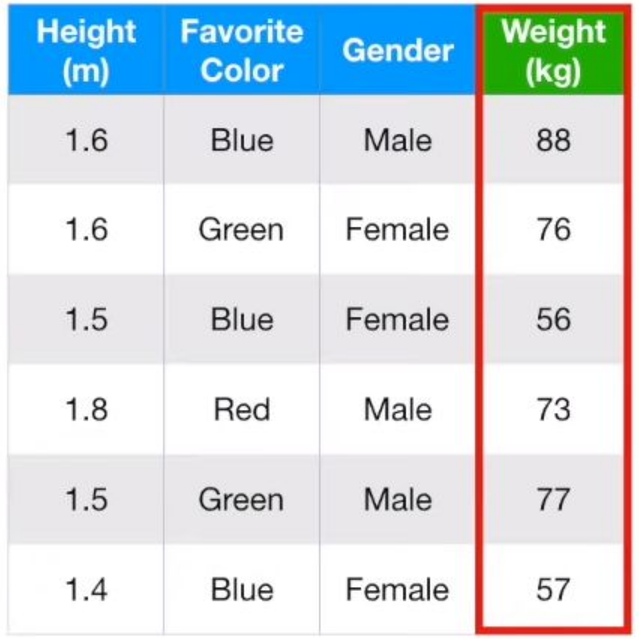


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
No	Yes	156	No
Yes	Yes	167	Yes
No	Yes	125	No
Yes	Yes	167	Yes
Yes	Yes	167	Yes
Yes	Yes	172	No
Yes	Yes	205	Yes
Yes	Yes	167	Yes

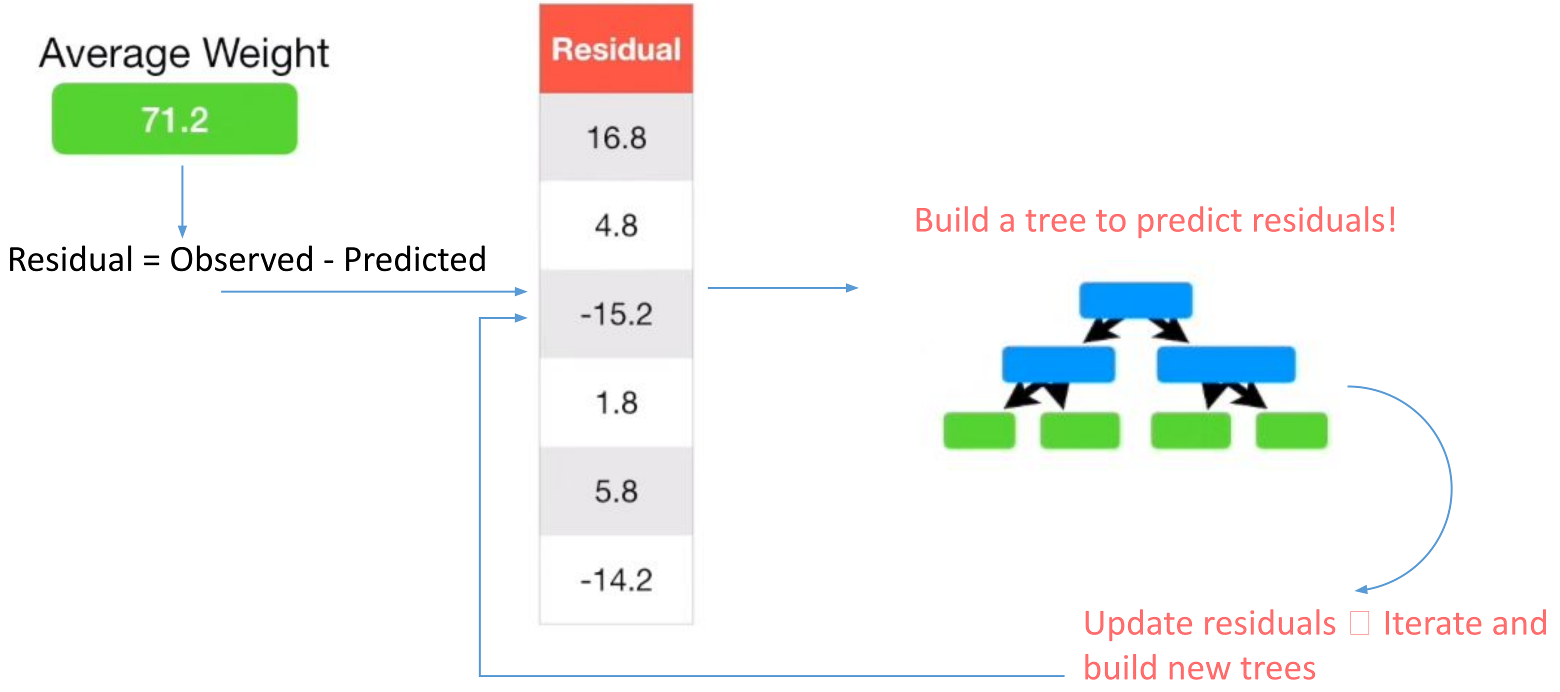
Boosting – *Gradient Boosting Machines (GBM)*

- Similar to AdaBoost
- AdaBoost uses *samples weights* to decrease errors for new decision trees
- Gradient Boosting uses *gradients* in the loss function for new decision trees

Instead of Stumps, GBM creates a node and a tree accordingly
□ Not a full tree, limited by number of leaves



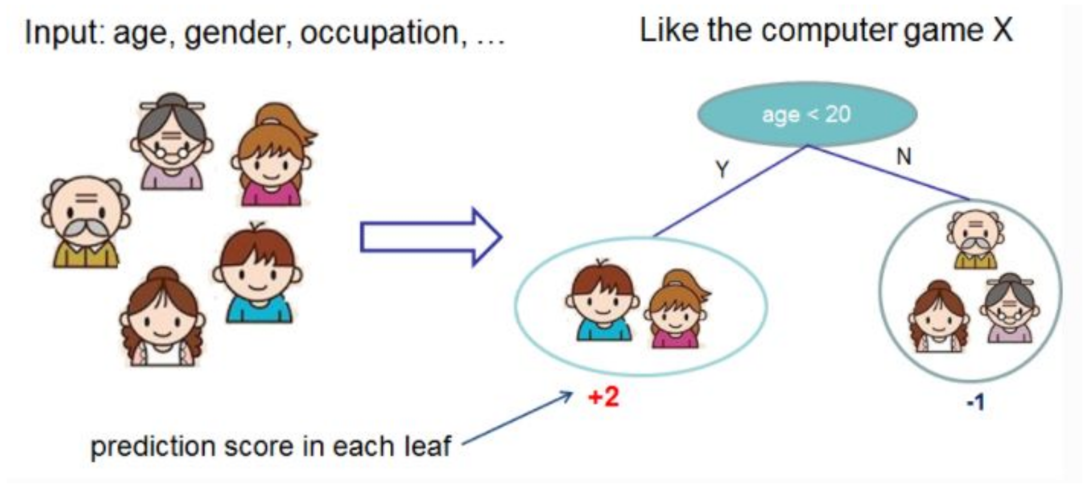
Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

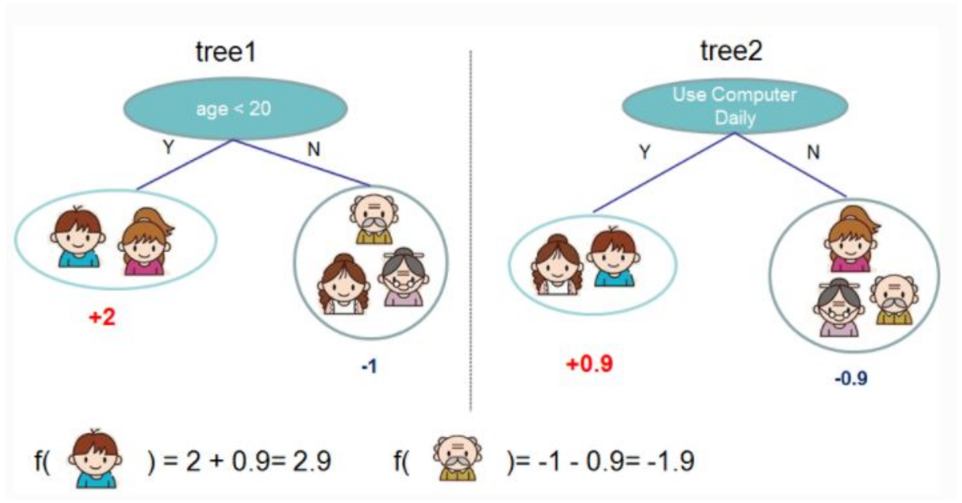

Average Weight
71.2
Residual = Observed - Predicted
Residual
16.8
4.8
-15.2
1.8
5.8
-14.2
Build a tree to predict residuals!
Update residuals □ Iterate and build new trees

XGBoost

Boosting – *XGBoost*

- eXtreme Gradient Boosting
- Similar to GBM, but employs *regularization*


Input: age, gender, occupation, …
Like the computer game X
age < 20
Y
N
+2
-1
prediction score in each leaf



Usually, a single tree is not strong enough to be used in practice. What is actually used is the ensemble model, which sums the prediction of *multiple trees* together.

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Boosting – *XGBoost*

- Objective Function

f( ) = 2 + 0.9= 2.9   f( )= -1 - 0.9= -1.9

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \omega(f_k)$$

We need to optimize this objective function

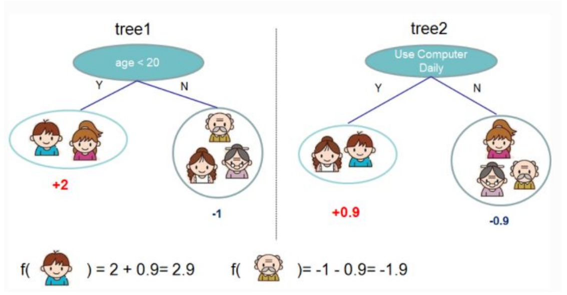
Complexity of the tree

Boosting – *XGBoost*

- Parameters of the tree *What to learn?*

$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \boxed{\sum_{i=1}^{t} \omega(f_i)}$$

Learn functions f_i , those containing the structure of the tree and leaf scores



$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Use an additive strategy: fix what we have learned, and add one new tree at a time

$$\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \omega(f_i) \\
&= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \omega(f_t) + \text{constant}
\end{aligned}$$

$$\begin{aligned}
g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\
h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})
\end{aligned}$$

Optimize this!

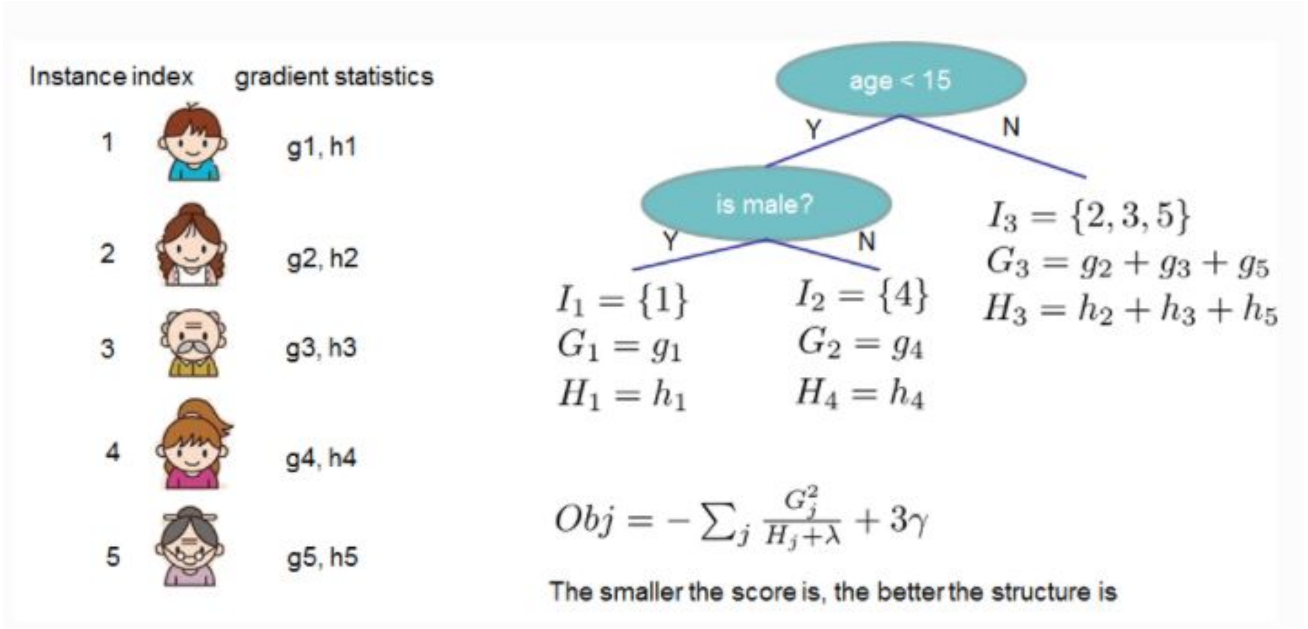
$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t)$$

Boosting – *XGBoost*

- Parameters of the tree *What to learn?*

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \omega(f_t)$$



Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5

$I_1 = \{1\}$, $G_1 = g_1$, $H_1 = h_1$

$I_2 = \{4\}$, $G_2 = g_4$, $H_4 = h_4$

$I_3 = \{2, 3, 5\}$, $G_3 = g_2 + g_3 + g_5$, $H_3 = h_2 + h_3 + h_5$

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

https://xgboost.readthedocs.io/en/latest/tutorials/model.html