# Boosting

# AdaBoost

# Boosting

- Can weak learners H be combined to generate a strong learner with low bias?
- Two modifications from bagging

  1. instead of a random sample of the training data, use a weighted sample to focus learning on most difficult examples.

  2. instead of combining classifiers with equal vote, use a weighted vote.

- Boosting: An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Unlike bagging, each record in boosting method is assigned a weight
  - Initially, all records are assigned equal weights
  - At each iteration, a new classifier (weak learner) is learned
  - Each record is reweighted to focus the system on data that the most recently learned classifier got wrong.

- Final classification based on weighted vote of weak learners

# Strong and Weak Learners

- Strong Learner
  - Produce a classifier with high accuracy
  - Relatively difficult to construct
  - E.g.: SVM, deep learning, etc

- Weak Learner
  - Produce a classifier which is more accurate than random guessing (accuracy>0.5)
  - Relatively easy to construct
  - E.g.: decision stump, etc

- Can a set of **weak learners** create a single **strong learner** ?
  - YES ! Boost weak classifiers to a strong learner

# Updating Weights of Data

- Using Different Data Distribution
  - Start with uniform weighting
  - During each step of learning
    - Increase weights of the examples which are not correctly learned by the weak learner
    - Decrease weights of the examples which are correctly learned by the weak learner

- Idea
  - Focus on difficult examples which are not correctly classified in the previous steps

# Updating Weights of Data

- 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

# Combine Weak Classifiers

- Weighted Voting
  - Construct strong classifier by weighted voting of the weak classifiers

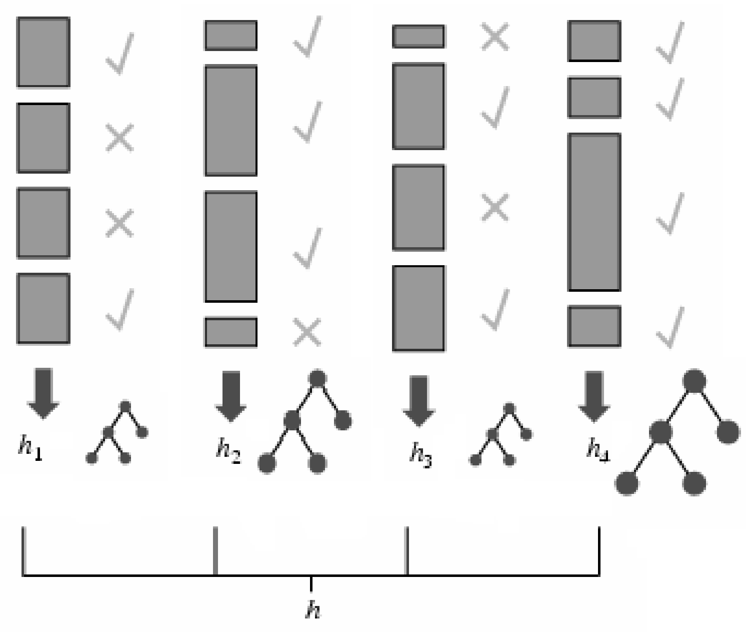
- Idea
  - Better weak classifier gets a larger weight
  - Iteratively add weak classifiers

# Boosting

- Each rectangle corresponds to an example, with weight proportional to its height.

- Crosses correspond to misclassified examples.

- Size of decision tree indicates the weight of that hypothesis in the final ensemble.



# Boosting

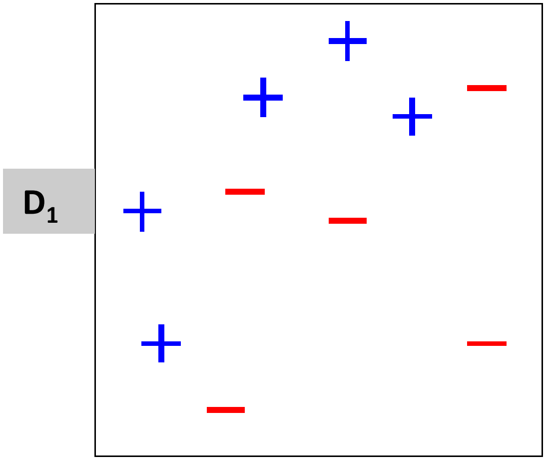
- M = number of weak learners to generate
- 1. Set same weight for all the examples (e.g., weight = 1);
- 2. For i = 1, … , M

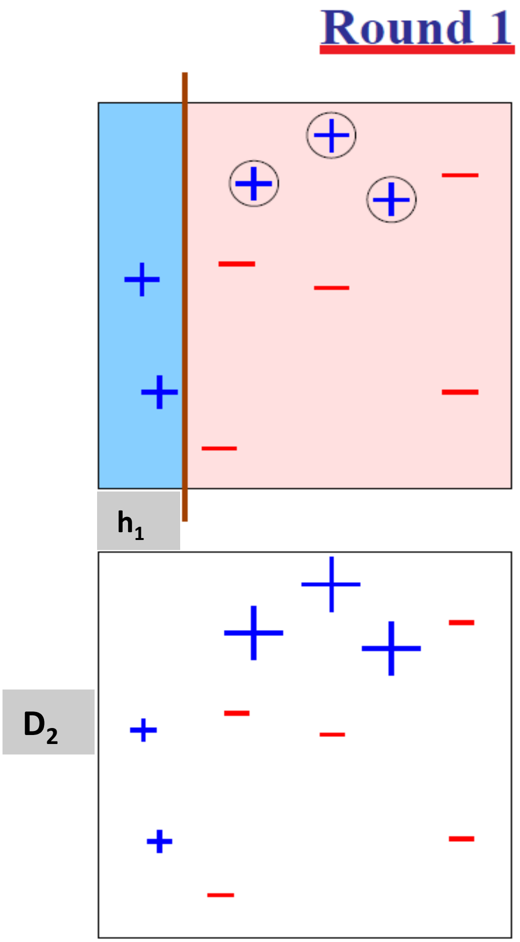
  2.1 Generate classifier $h_i$ .

  2.2 Recompute weights of examples from the predictions of $h_i$

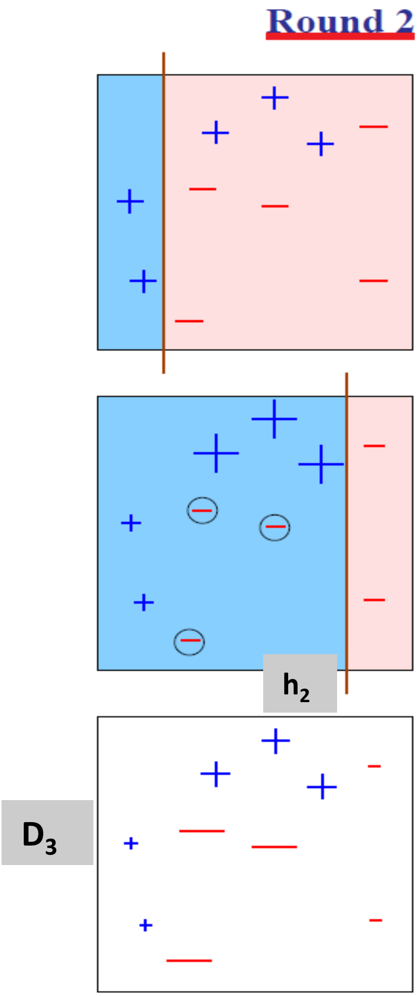
  2.3 Recompute the weight of classifier $h_i$
- 3. Weighted majority combination of all M classifiers (weights according to how well it performed on the training set).
- Many variants depending on how to set the weights and how to combine the classifiers.

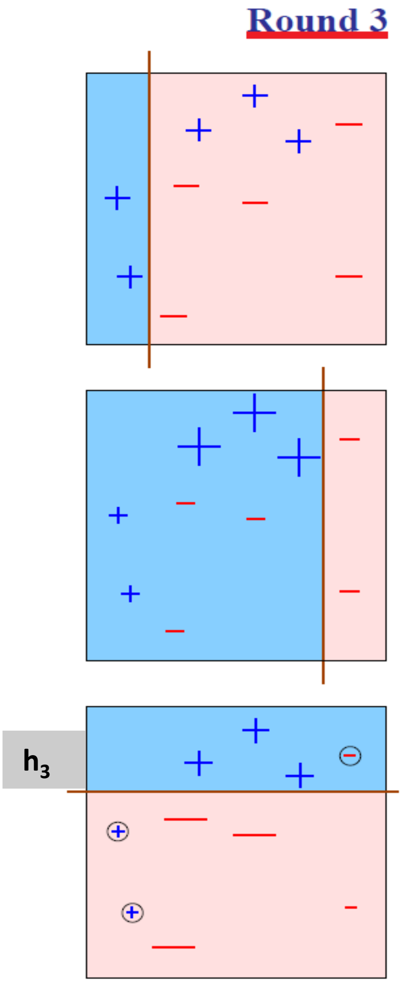
# A Toy Example



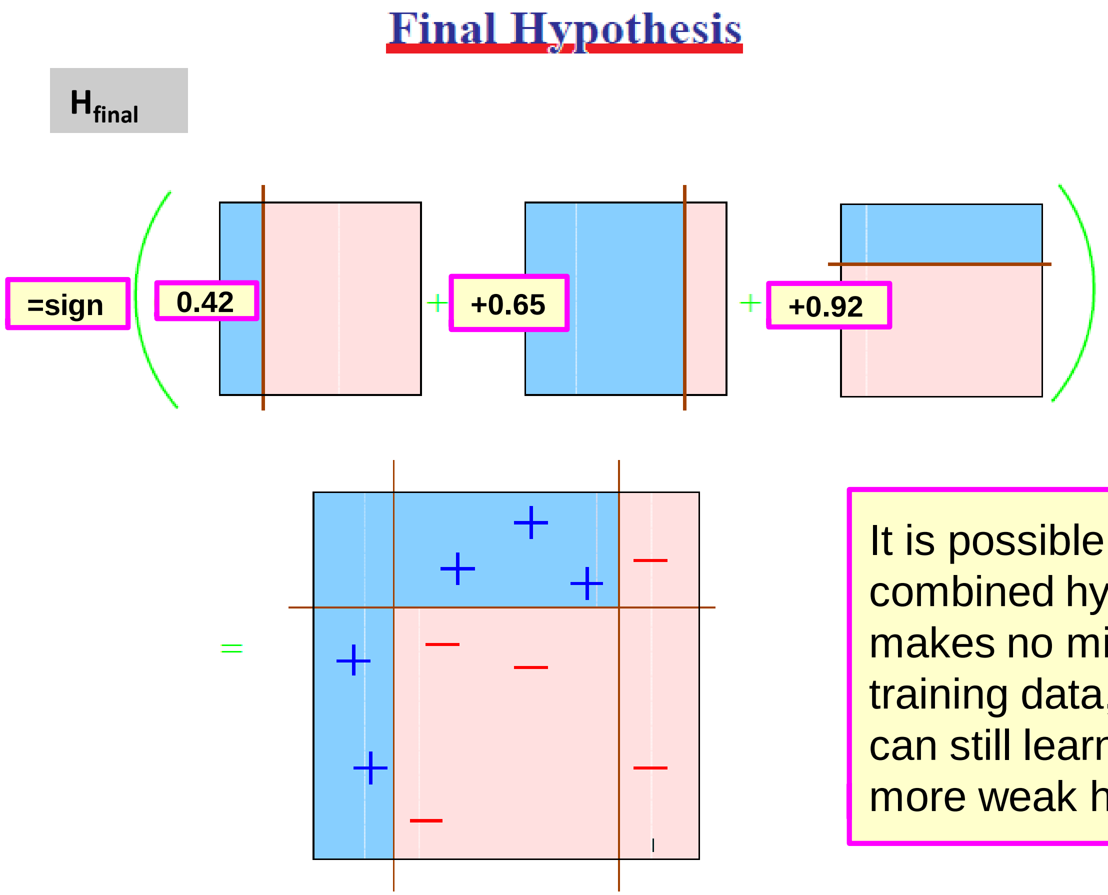



# A Toy Example



# A Toy Example




It is possible that the combined hypothesis makes no mistakes on the training data, but boosting can still learn, by adding more weak hypotheses.

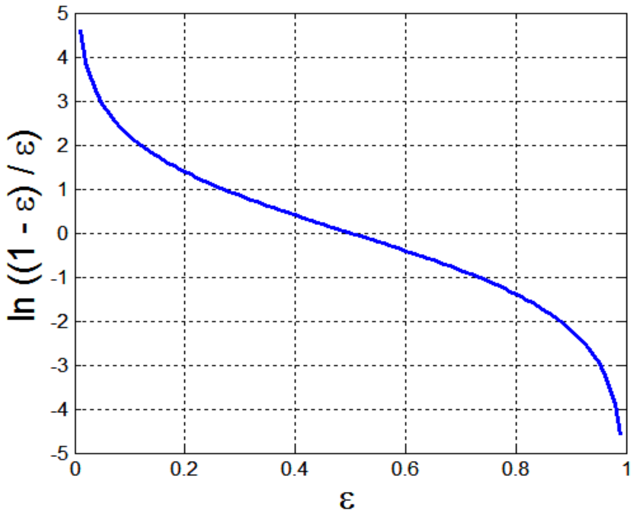
# AdaBoost

- Base classifiers: $C_1$, $C_2$, …, $C_T$
- Error rate: ($\varepsilon_i$: error of classifier *i*)

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

  – $N$ : number of data
  – $\delta(p) = 1$ if *p* is true, and
  =o otherwise
- Importance of a classifier:

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




AdaBoost by Yoav Freund and Robert E. Schapire, 1996

# AdaBoost

- Weight update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} e^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ e^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases} \qquad \epsilon_j < 0.5 \ \& \ \alpha_j > 0$$

  – $w_i^{(j)}$ : the weight assigned to example $(x_i, y_i)$ during *j*-th round
  – $Z_j$ : normalization factor so that $\sum_i w_i^{(j+1)} = 1$

- If any intermediate rounds produce error rate($\epsilon_j$) > 0.5, the weights are reverted back to 1/n and the resampling procedure is repeated

- Classification: weighted vote according to $\alpha_j$

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

# Illustrating AdaBoost

Original data

| X | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

Boosting round 1

| X | 0.1 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 | 0.7 | 0.7 | 0.8 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 of records

| 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 | 0.31 | 0.31 | 0.31 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 3 | 0.03 | 0.03 | 0.03 | 0.23 | 0.23 | 0.23 | 0.23 | 0.01 | 0.01 | 0.01 |

# Illustrating AdaBoost

| Round | Split Point | Left Class | Right Class | $\alpha$ |
|---|---|---|---|---|
| 1 | 0.75 | -1 | 1 | 1.738 |
| 2 | 0.05 | 1 | 1 | 2.778 |
| 3 | 0.3 | 1 | -1 | 4.119 |

| 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 | 0.40 | 0.40 | 0.40 |
| Sign | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| True Class | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

# Pros and Cons of AdaBoost

- Advantages
  - Very simple to implement
  - Fairly good generalization

- Disadvantages
  - Suboptimal solution
  - Sensitive to noisy data and outliers

- AdaBoost with decision trees has been referred to as “the best off-the-shelf classifier”. However,
  - If weak learners are actually quite strong (i.e., error gets small very quickly), boosting might not help
  - If hypotheses too complex, test error might be much larger than training error

# Gradient Boosting

# Gradient Boosting

- Trees are added one at a time, and existing trees in the model are not changed.
- A gradient descent procedure is used to minimize the loss when adding trees.
- Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network.
- In gradient boosting, instead of updating parameters, we must add a tree to the model that reduces the loss (i.e. follow the gradient).
- We modify the parameters of the new tree so that it moves in the right direction by (reducing the residual loss). In other words, we train a decision tree to reduce the residual loss.
- The output for the new tree is then added to the output of the existing sequence of trees

# Gradient Boosting

- Create ensemble classifier

$$H_T(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x})$$

- This ensemble classifier is built in an iterative fashion.
- In iteration $t$ we add the classifier $\alpha_t h_t(\vec{x})$ to the ensemble.
- At test time we evaluate all classifier and return the weighted sum.

# Gradient Boosting

- The process of constructing such an ensemble in a step-wise fashion is very similar to gradient descent.
- However, instead of updating the model parameters in each iteration, we add functions to our ensemble.
- Error function:

$$\sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

- We add a new function $h_t$ each time.

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = h_1(x_i) = \hat{y}_i^{(0)} + h_1(x_i)$$
$$\hat{y}_i^{(2)} = h_1(x_i) + h_2(x_i) = \hat{y}_i^{(1)} + h_2(x_i)$$
...
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} h_k(x_i) = \hat{y}_i^{(t-1)} + h_t(x_i)$$

# Gradient Boosting

- We want to optimize the error function:

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$$

- Assume we have already finished $t-1$ iterations and already have an ensemble classifier $\hat{y}_i^{(t-1)}$.
- Now in iteration t, we want to add one more weak learner $h_t$ to the ensemble.
- Performance at round $t$ is

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} h_k(x_i) = \hat{y}_i^{(t-1)} + h_t(x_i)$$

- To this end, we search for the weak learner $h_t$ that minimizes the loss the most.
- How do we decide which $h_t$ to add?

- Find $h_t$ that minimizes the following

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + h_t(x_i))$$

# Gradient Boosting

$$L = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + h_t(x_i))$$

- Take Taylor expansion of the error function
  - Recall: $f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f(x)\Delta x^2 + \cdots$ *(we will use first-degree term only)*
  - Define:

$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i h_t(x_i) \right] \approx$$
$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [g_i h_t(x_i)] + \text{const}$$

where

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$$

- You can use any other differentiable and convex loss function $l$, and the solution for your next weak learner $h$ will always be the regression tree minimizing the squared loss

# Gradient Boosting

- For simplicity, suppose the error function is squared error.

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}} = \frac{\partial \left(\hat{y}_i^{(t-1)} - y_i\right)^2}{\partial \hat{y}_i^{(t-1)}} = 2(\hat{y}_i^{(t-1)} - y_i)$$

- Therefore, the error function is

$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) h_t(x_i)\right]$$

- Suppose $r_i = \hat{y}_i^{(t-1)} - y_i$
- We need a function(weak learner) $A$ such that

$$A(\{(x_1, r_1), \dots, (x_n, r_n)\}) = \underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [r_i \ h_t(x_i)]$$

- In order to make progress, this $h_t$ does not have to be great. We still make progress as long as $\sum_{i=1}^{n} [r_i \ h_t(x_i)] < 0$.

# (*) Minimizing Dot Product

$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [r_i\, h_t(x_i)]$$

- To minimize the inner product of a given vector $v$ with another vector $u$, you want to find a vector $u$ that is as close to orthogonal to $v$ as possible. Since the inner product is given by:

$$v \cdot u = \|v\|\|u\| \cos\theta$$

where $\theta$ is the angle between $v$ and $u$, the inner product is minimized when $\cos\theta = -1$. This corresponds to $\theta = \pi$, meaning $u$ is in the opposite direction of $v$.

- Therefore, the vector $u$ that minimizes the inner product with $v$:

$$u = -\alpha v$$

where $\alpha$ is a positive scalar that scales $u$ (it can be any positive number depending on the magnitude you want for $u$).

- In short, to minimize the inner product, you should choose $u$ as a vector pointing in the exact opposite direction of $v$, scaled by any positive factor.

# Gradient Boosting

$$\underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [r_i \, h_t(x_i)]$$

- From previous slide, in order to minimize above formula, we know that the prediction of $h_t(x_i)$ should be the opposite value of $r_i$

- For example, if $r = [3, -2, 6]$, then the prediction of $h_t$ should be $[-3, 2, -6]$. (suppose there are three data points)

- In other words, when we build a new weak learner, $(-1) * r_i$ becomes the target value of the new weak learner.

# Gradient Boosting Algorithm

* Regression with squared error

**Input:** $\{(x_i, y_i)\}$: initial training data; A: weak classifier;
T total number of iterations; $H^0$: ensemble at step $t$

$h_0 = \overline{y}_i$ # mean of $y_i$

$H^{(0)} = \{h_0\}$

**for** $t = 1; T - 1$ **do**

$r_i = \hat{y}_i^{(t-1)} - y_i$ # for each data i, compute residual

# with $r_i$ as the target values, generate next weak classifier A

$h_t = A(\{(x_1, r_1), \dots, (x_n, r_n)\}) = \underset{h_t}{\operatorname{argmin}} \sum_{i=1}^{n} [r_i \, h_t(x_i)]$

**if** $\sum_{i=1}^{n} [r_i \, h_t(x_i)] < 0$ **then**

$H^{(t)} = H^{(t-1)} + h_t$

**else**

**return** $H^{(t)}$

**end if**

**end for**

**return** $H^{(T)}$

# XGBoost Algorithm

- XGBoost stands for "eXtreme Gradient Boosting" and is a specific implementation of the gradient boosting algorithm.

- One of the state-of-the-art methods for tabular data

- Some differences between the two:
  - Algorithm: XGBoost uses Newton-Raphson in function space, while gradient boosting uses gradient descent.
  - Regularization: XGBoost uses advanced regularization (L1 and L2) to improve model generalization.
  - Parallelization: XGBoost is faster to train than gradient boosting and can be parallelized across clusters.
  - Missing Data: XGBoost has its own in-built missing data handler, whereas GBM doesn't.

# Gradient Boosting Example

| Age | Sft. | Location | Price |
|---|---|---|---|
| 5 | 1500 | 5 | 480 |
| 11 | 2030 | 12 | 1090 |
| 14 | 1442 | 6 | 350 |
| 8 | 2501 | 4 | 1310 |
| 12 | 1300 | 9 | 400 |
| 10 | 1789 | 11 | 500 |

| Age | Sft. | Location | Price | Average_Price |
|---|---|---|---|---|
| 5 | 1500 | 5 | 480 | 688 |
| 11 | 2030 | 12 | 1090 | 688 |
| 14 | 1442 | 6 | 350 | 688 |
| 8 | 2501 | 4 | 1310 | 688 |
| 12 | 1300 | 9 | 400 | 688 |
| 10 | 1789 | 11 | 500 | 688 |

$h_1$ is the mean of targer value(Price)

$$\frac{480+1090+350+1310+400+500}{6} = 688$$

'Average_Price' is the prediction of $h_1$

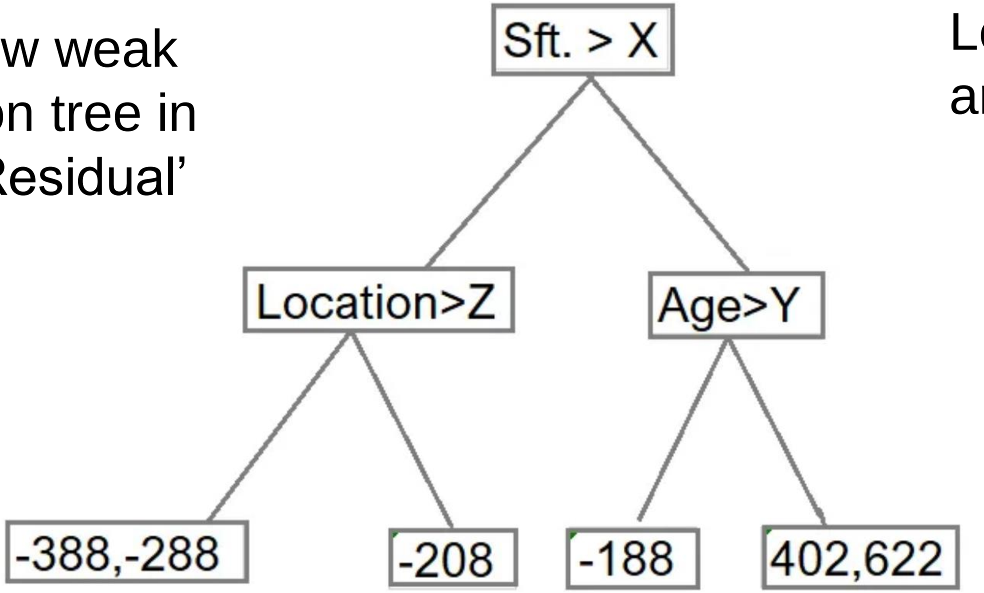
# Gradient Boosting Example

| Age | Sft. | Location | Price | Average_Price | Residual |
|---|---|---|---|---|---|
| 5 | 1500 | 5 | 480 | 688 | -208 |
| 11 | 2030 | 12 | 1090 | 688 | 402 |
| 14 | 1442 | 6 | 350 | 688 | -338 |
| 8 | 2501 | 4 | 1310 | 688 | 622 |
| 12 | 1300 | 9 | 400 | 688 | -288 |
| 10 | 1789 | 11 | 500 | 688 | -188 |

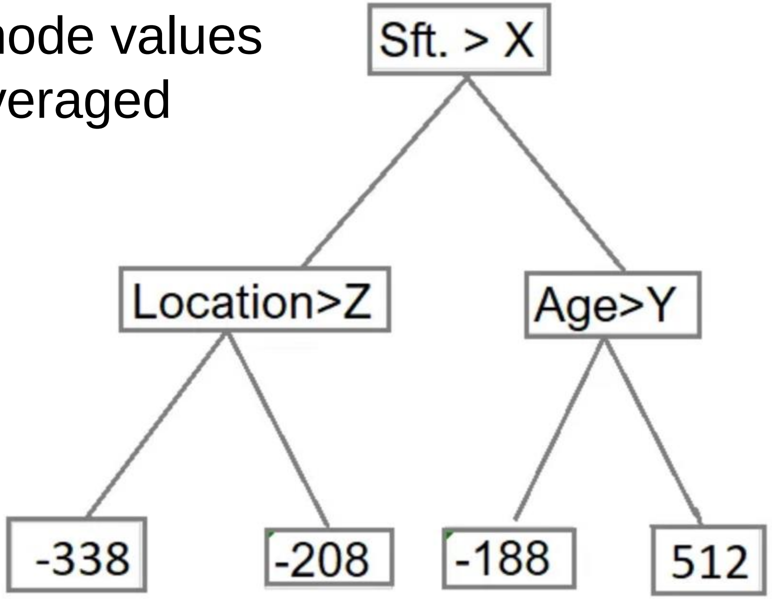
From the prediction of $h_1$, we compute 'Residual'

Residual = target value – predicted value

We construct a new weak learner $h_2$ (decision tree in this case) using 'Residual' as the target



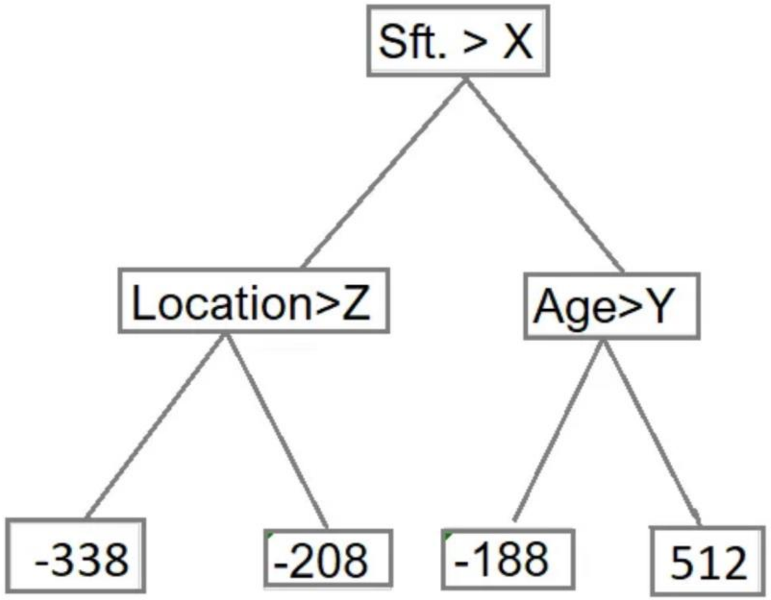

Leaf node values are averaged



# Gradient Boosting Example

Using this decision tree, we compute 'New Residual'

| Age | Sft. | Location | Price | Average_Price | Residual |
|---|---|---|---|---|---|
| 5 | 1500 | 5 | 480 | 688 | -208 |
| 11 | 2030 | 12 | 1090 | 688 | 402 |
| 14 | 1442 | 6 | 350 | 688 | -338 |
| 8 | 2501 | 4 | 1310 | 688 | 622 |
| 12 | 1300 | 9 | 400 | 688 | -288 |
| 10 | 1789 | 11 | 500 | 688 | -188 |




| Age | Sft. | Location | Price | Average_Price | Residual | New Residual |
|---|---|---|---|---|---|---|
| 5 | 1500 | 5 | 480 | 688 | -208 | -187.2 |
| 11 | 2030 | 12 | 1090 | 688 | 402 | 350.8 |
| 14 | 1442 | 6 | 350 | 688 | -338 | -304.2 |
| 8 | 2501 | 4 | 1310 | 688 | 622 | 570.8 |
| 12 | 1300 | 9 | 400 | 688 | -288 | -254.1 |
| 10 | 1789 | 11 | 500 | 688 | -188 | -169.2 |

Repeat this process using 'New Residual'