Lecture 11: 17 February, 2026

Madhavan Mukund

`https://www.cmi.ac.in/~madhavan`

Data Mining and Machine Learning
January–April 2026

Limitations of classification models

Recall

- Bias : Expressiveness of model limits classification

- Variance: Variation in model based on sample of training data

Overcoming limitations

- Bagging is an effective way to overcome high variance

  - Ensemble models

    - Sequence of models based on independent bootstrap samples
    - Use voting to get an overall classifier

- How can we cope with high bias?

Dealing with bias

- A biased model always makes mistakes
    - Build an ensemble of models to average out mistakes

- Mistakes should be compensated across models in the ensemble
    - How to build a sequence of models, each biased a different way?
    - Again, we assume we have only one set of training data

Boosting

- Build a sequence of weak classifiers M_1, M_2, ..., M_n on inputs D_1, D_2, ..., D_n
  - A weak classifier is any classifier that has error rate strictly below 50%

- Each D_i is a weighted variant of original training data D
  - Initially all weights equal, D_1
  - Going from D_i to D_{i+1} : increase weights where M_i makes mistakes on D_i
  - M_{i+1} will compensate for errors of M_i

- Also, each model M_i gets a weight α_i based on its accuracy on D_i

- Ensemble output
  - Individual classification outcomes are $\{-1, +1\}$
  - Unknown input x: ensemble outcome is weighted sum $\sum_{i=1}^{n} \alpha_i M_i(x)$
  - Check if weighted sum is negative/positive

The boosting algorithm — Adaboost

- Initially, all data items have equal weight

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $f_t \leftarrow$ BaseLearner(D_t);
4. $e_t \leftarrow \sum\limits_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5.   **if** $e_t > \frac{1}{2}$ **then**
6. $k \leftarrow k - 1$;
7.   exit-loop
8.   **else**
9. $\beta_t \leftarrow e_t / (1 - e_t)$;
10. $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

- Initially, all data items have equal weight

- Build a new model and compute its weighted error

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $\quad f_t \leftarrow \text{BaseLearner}(D_t)$;
4. $\quad e_t \leftarrow \sum\limits_{i:\, f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5. $\quad$ **if** $e_t > \frac{1}{2}$ **then**
6. $\quad\quad k \leftarrow k - 1$;
7. $\quad\quad$ exit-loop
8. $\quad$ **else**
9. $\quad\quad \beta_t \leftarrow e_t / (1 - e_t)$;
10. $\quad\quad D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $\quad\quad D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

- Initially, all data items have equal weight

- Build a new model and compute its weighted error

- Discard if error rate is above 50%

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $f_t \leftarrow$ BaseLearner(D_t);
4. $e_t \leftarrow \displaystyle\sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5.   **if** $e_t > \frac{1}{2}$ **then**
6. $k \leftarrow k - 1$;
7.   exit-loop
8.   **else**
9. $\beta_t \leftarrow e_t / (1 - e_t)$;
10. $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

The boosting algorithm — Adaboost

- Initially, all data items have equal weight

- Build a new model and compute its weighted error

- Discard if error rate is above 50%

- Damping factor — reduce weight of correct inputs

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $f_t \leftarrow$ BaseLearner(D_t);
4. $e_t \leftarrow \sum\limits_{i:f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5.   **if** $e_t > \frac{1}{2}$ **then**
6. $k \leftarrow k - 1$;
7.   exit-loop
8.   **else**
9. $\beta_t \leftarrow e_t / (1 - e_t)$;
10. $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

The boosting algorithm — Adaboost

- Initially, all data items have equal weight

- Build a new model and compute its weighted error

- Discard if error rate is above 50%

- Damping factor — reduce weight of correct inputs

- Reweight data items and normalize

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $f_t \leftarrow$ BaseLearner(D_t);
4. $e_t \leftarrow \displaystyle\sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5.   **if** $e_t > \frac{1}{2}$ **then**
6. $k \leftarrow k - 1$;
7.   exit-loop
8.   **else**
9. $\beta_t \leftarrow e_t / (1 - e_t)$;
10. $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

The boosting algorithm — Adaboost

- Initially, all data items have equal weight

- Build a new model and compute its weighted error

- Discard if error rate is above 50%

- Damping factor — reduce weight of correct inputs

- Reweight data items and normalize

- Final classifier

$$f_{\text{final}}(x) = \arg\max_{y \in Y} \sum_{t:f_t(x)=y} \log \frac{1}{\beta_t}$$

AdaBoost(D, Y, BaseLeaner, k)

1. Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
2. **for** $t = 1$ to k **do**
3. $\quad f_t \leftarrow$ BaseLearner(D_t);
4. $\quad e_t \leftarrow \sum_{i:f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
5. $\quad$ **if** $e_t > \frac{1}{2}$ **then**
6. $\quad\quad k \leftarrow k - 1$;
7. $\quad\quad$ exit-loop
8. $\quad$ **else**
9. $\quad\quad \beta_t \leftarrow e_t / (1 - e_t)$;
10. $\quad\quad D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
11. $\quad\quad D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

The boosting algorithm — Adaboost

- Each M_i could be a different type of model

- Start with N weak classifiers

- Initially all data items have equal weight, select M_1 as model with lowest error rate among N candidates

- Inductively, assume we have selected $M_1, \ldots M_j$, with model weights $\alpha_1, \ldots, \alpha_j$, and dataset is updated with new weights as D_{j+1}
    - Pick model with lowest error rate on D_{j+1} as M_{j+1}
    - Calculate α_{j+1} based on error rate of M_{j+1}
    - Reweight all training data based on error rate of M_{j+1}

- Note that same model M may be picked in multiple iterations, assigned different weights α
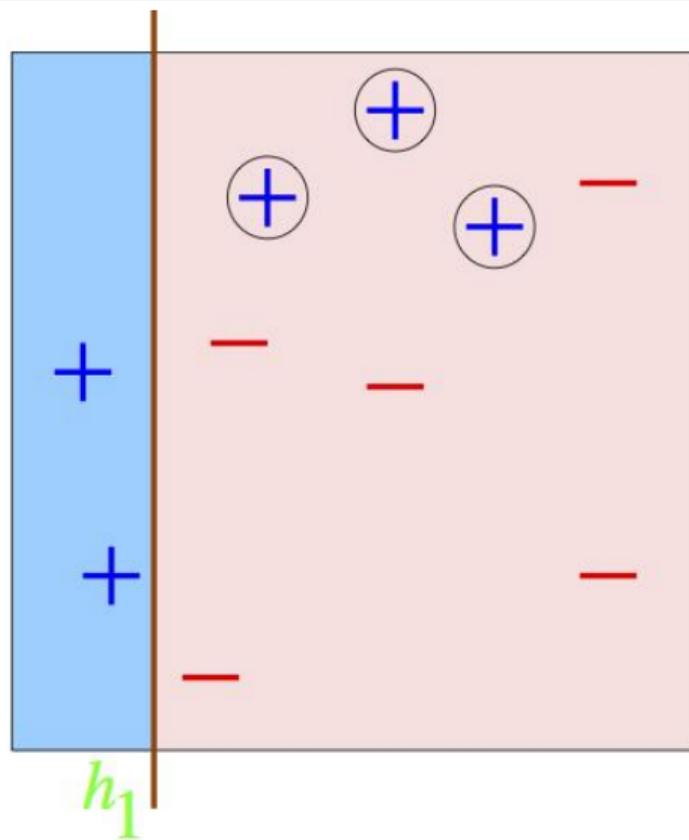
Boosting: An example

- Weak classifiers are horizontal and vertical lines

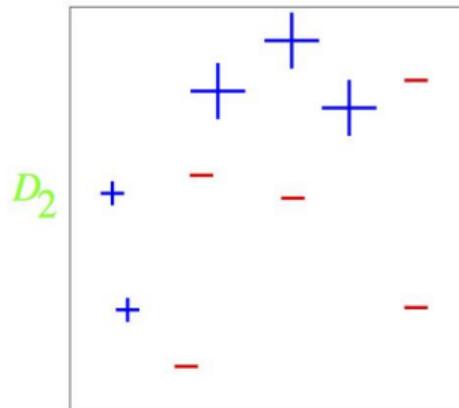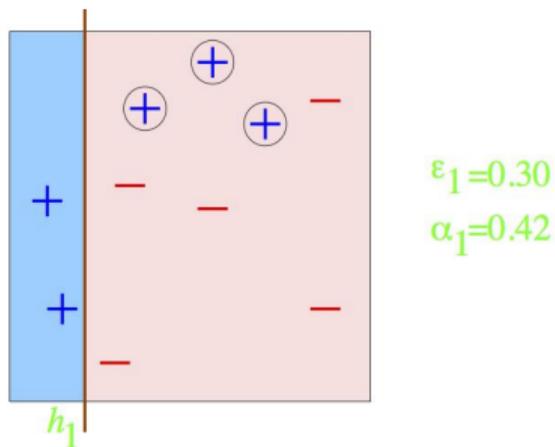- Initial training data has equal weights

Boosting: An example
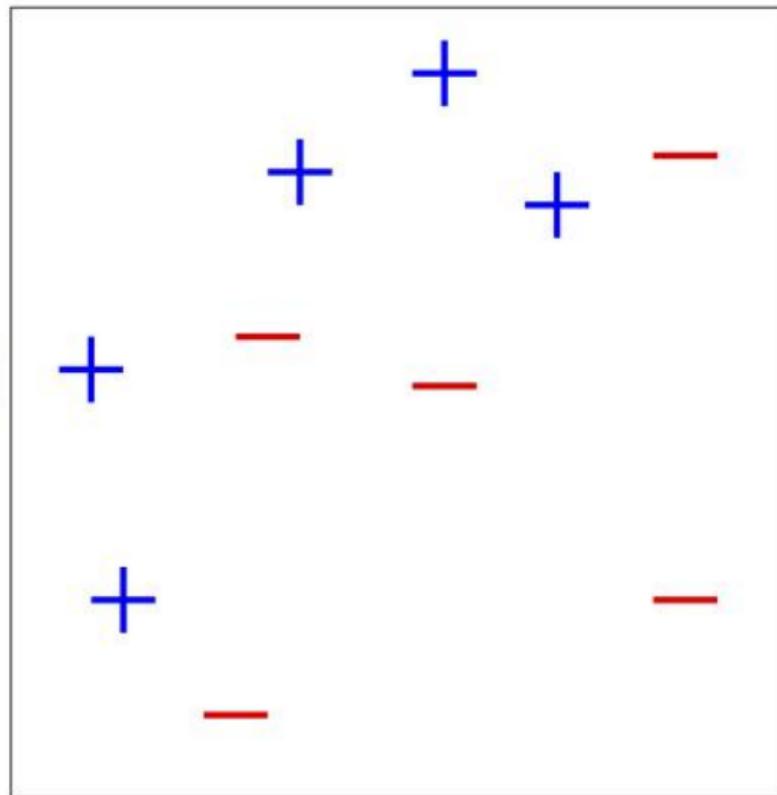
- Weak classifiers are horizontal and vertical lines

- Initial training data has equal weights

- First separator: vertical line
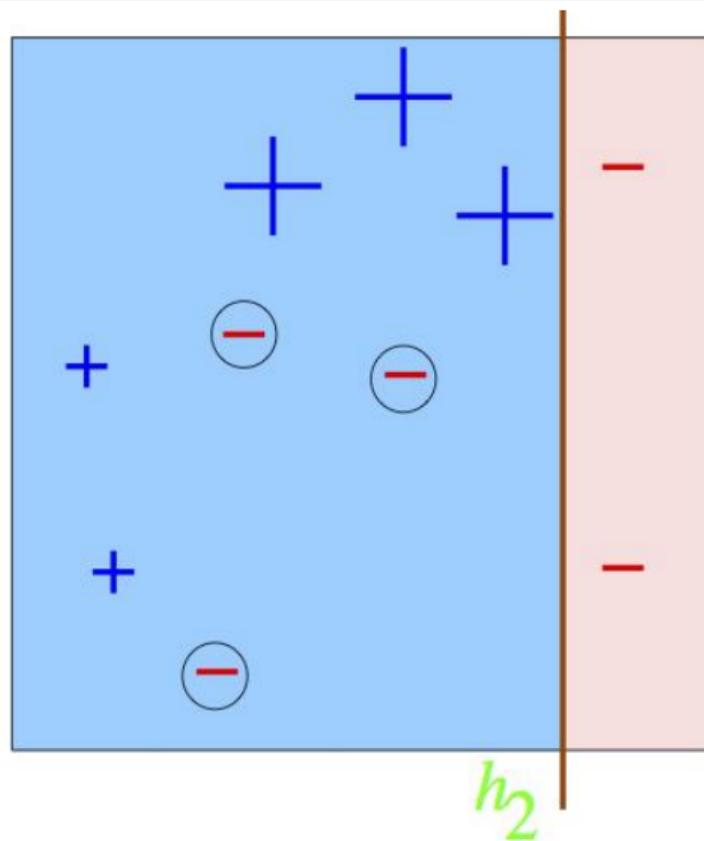

$$\varepsilon_1$$

$$\alpha_1$$

Boosting: An example

- Weak classifiers are horizontal and vertical lines

- Initial training data has equal weights

- First separator: vertical line
  - Increase weight of misclassified inputs

$\varepsilon_1 = 0.30$
$\alpha_1 = 0.42$

h_1

D_2

Boosting: An example

- Weak classifiers are horizontal and vertical lines

- Initial training data has equal weights

- First separator: vertical line
  - Increase weight of misclassified inputs

- Second separator: vertical line
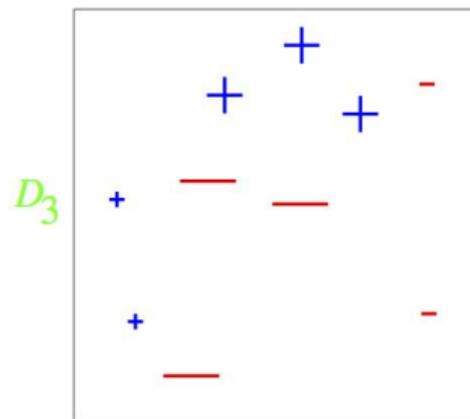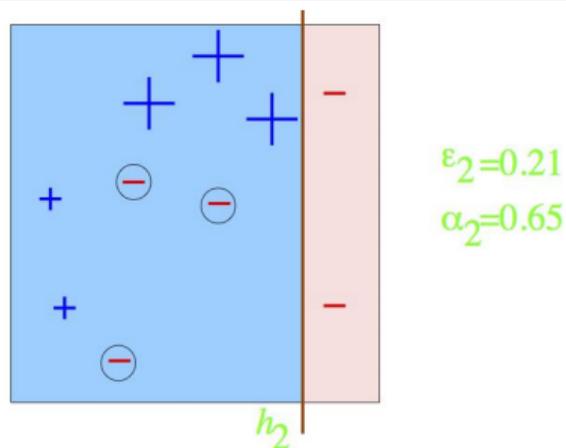
ε_2

α_2

h_2

Boosting: An example

- Weak classifiers are horizontal and vertical lines

- Initial training data has equal weights

- First separator: vertical line
  - Increase weight of misclassified inputs

- Second separator: vertical line
  - Increase weight of misclassified inputs

$\varepsilon_2 = 0.21$
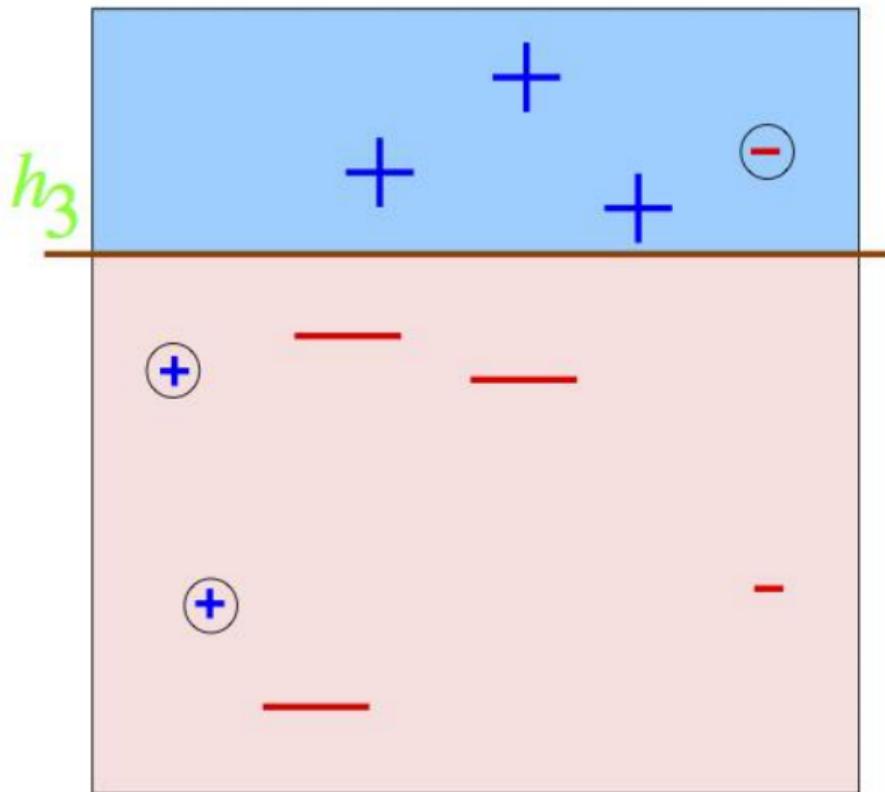$\alpha_2 = 0.65$

h_2

D_3

Boosting: An example

- Weak classifiers are horizontal and vertical lines

- Initial training data has equal weights

- First separator: vertical line
  - Increase weight of misclassified inputs

- Second separator: vertical line
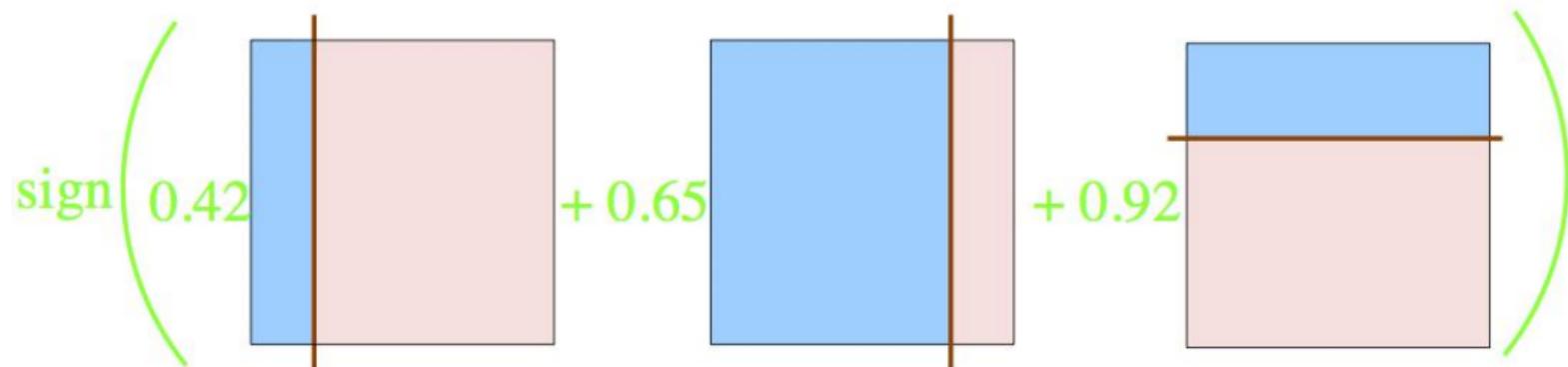 - Increase weight of misclassified inputs

- Third separator: horizontal line

- Final classifier is weighted sum of three weak classifiers

$$\text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \right)$$

- Pictorially
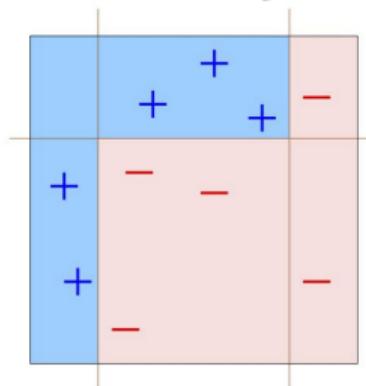
Gradient Boosting

- AdaBoost uses weights to build new weak learners that compensate for earlier errors

- Gradient boosting follows a different approach
    - Shortcomings of the current model are defined in terms of gradients
    - Gradient boosting = Gradient descent + boosting

Gradient Boosting for Regression

- Training data (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)

- Fit a model $F(x)$ to minimize square loss

- The model F we build is good, but not perfect
  - $y_1 = 0.9$, $F(x_1) = 0.8$
  - $y_2 = 1.3$, $F(x_2) = 1.4$
  - ...

- Add an additional model h, so that new prediction is $F(x) + h(x)$

- What should h look like?

- For each x_i, want $F(x_i) + h(x_i) = y_i$

- $h(x_i) = y_i - F(x_i)$

- Fit a new model h (typically a regression tree) to the residuals $y_i - F(x_i)$

- If $F + h$ is not satisfactory, build another model h' to fit residuals $y_i - [F(x_i) + h(x_i)]$

- Why should this work?

Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i}$$
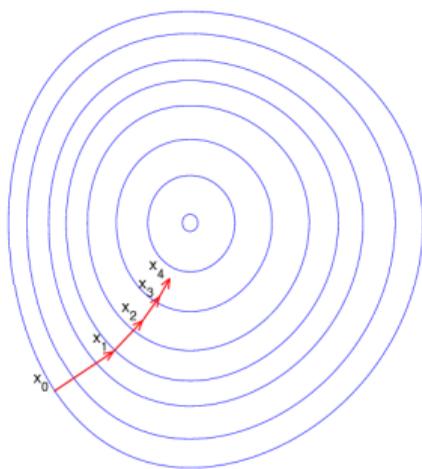
- Individual loss:
  $L(y, F(x)) = (y - F(x))^2/2$

- Minimize overall loss:
  $$J = \sum_i L(y_i, F(x_i))$$

- $\dfrac{\partial J}{\partial F(x_i)} = F(x_i) - y$

- Residual $y_i - F(x_i)$ is negative gradient

- Fitting h to residual is same as fitting h to negative gradient

- Updating F using residual is same as updating F based on negative gradient

Regression Trees

- Predict age based on given attributes
- Build a regression tree using CART algorithm

Person ID	Age	Likes Gardening	Plays Video Games	Likes Hats
1	13	FALSE	TRUE	TRUE
2	14	FALSE	TRUE	FALSE
3	15	FALSE	TRUE	FALSE
4	25	TRUE	TRUE	TRUE
5	35	FALSE	TRUE	TRUE
6	49	TRUE	FALSE	FALSE
7	68	TRUE	TRUE	TRUE
8	71	TRUE	FALSE	FALSE
9	73	TRUE	FALSE	TRUE

{13,14,15,25,35,49,68,71,73}

Gardening=F   Gardening=T

{13,14,15,35}   {25,49,68,71,73}

Hats=F   Hats=T   VideoG=F   VideoG=T

{14,15}   {13,35}   {49,71,73}   {25,68}

Person ID	Age	Likes Gardening	Plays Video Games	Likes Hats
1	13	FALSE	TRUE	TRUE
2	14	FALSE	TRUE	FALSE
3	15	FALSE	TRUE	FALSE
4	25	TRUE	TRUE	TRUE
5	35	FALSE	TRUE	TRUE
6	49	TRUE	FALSE	FALSE
7	68	TRUE	TRUE	TRUE
8	71	TRUE	FALSE	FALSE
9	73	TRUE	FALSE	TRUE

- **LikesHats** seems irrelevant, yet pops up
- Can we do better?

{13,14,15,25,35,49,68,71,73}

Gardening=F   Gardening=T

{13,14,15,35}   {25,49,68,71,73}

Tree 1

{-6.25,-5.25,-4.25,-32.2,15.75,-8.2,10.8,13.8,15.8}

VideoGames=F   VideoGames=T

{-8.2,13.8,15.8}   {-6.25,-5.25,-4.25,-32.2,15.75,10.8}
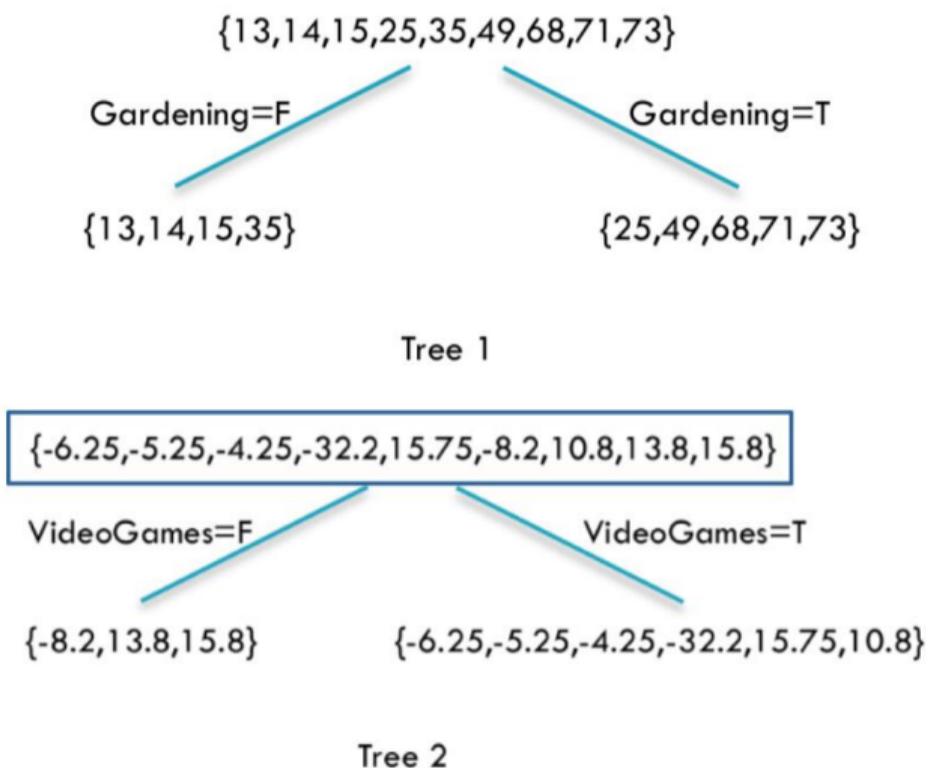
Tree 2

PersonID	Age	Tree1 Prediction	Tree1 Residual
1	13	19.25	-6.25
2	14	19.25	-5.25
3	15	19.25	-4.25
4	25	57.2	-32.2
5	35	19.25	15.75
6	49	57.2	-8.2
7	68	57.2	10.8
8	71	57.2	13.8
9	73	57.2	15.8

Residuals

{13,14,15,25,35,49,68,71,73}

Gardening=F | Gardening=T

{13,14,15,35}   {25,49,68,71,73}

Tree 1

{-6.25,-5.25,-4.25,-32.2,15.75,-8.2,10.8,13.8,15.8}

VideoGames=F | VideoGames=T

{-8.2,13.8,15.8}   {-6.25,-5.25,-4.25,-32.2,15.75,10.8}
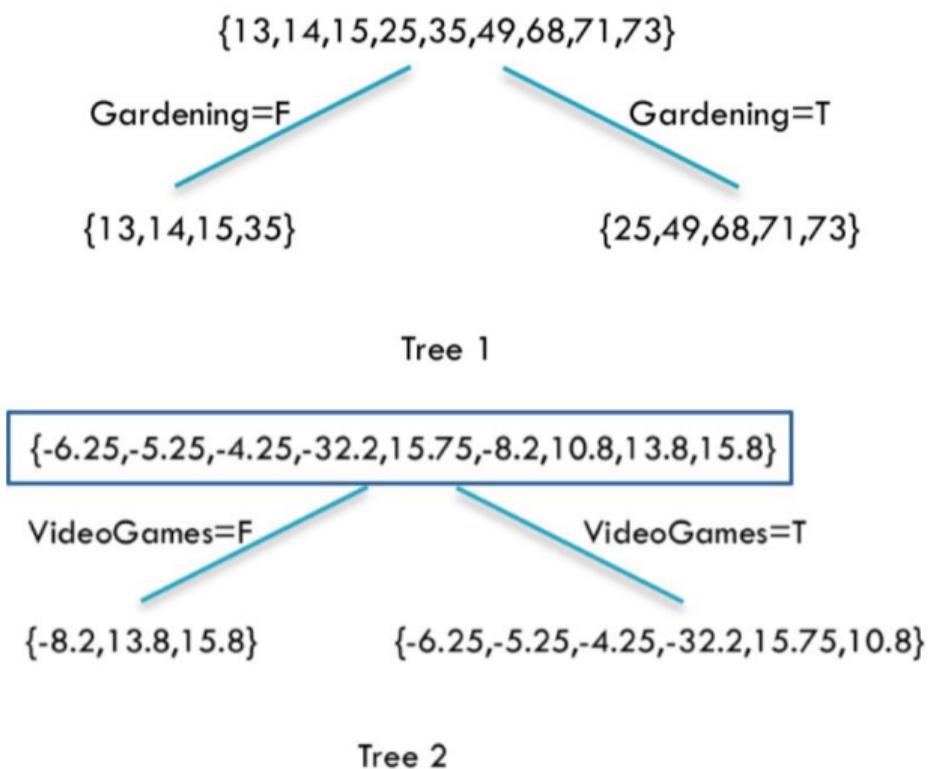
Tree 2

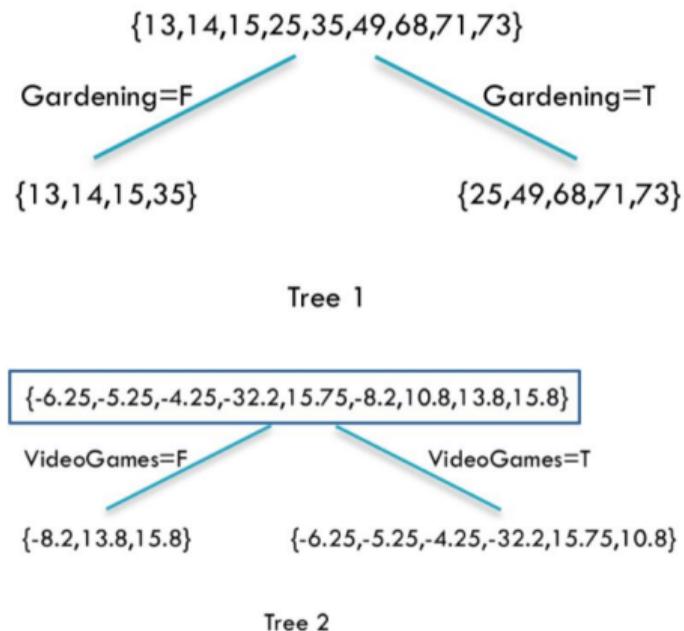| Per son ID	A g e	Tree1 Prediction	Tree1 Residual	Tree2 Prediction	Combined	Final Residual
1	13	19.25	-6.25	-3.567	15.68	-2.683
2	14	19.25	-5.25	-3.567	15.68	-1.683
3	15	19.25	-4.25	-3.567	15.68	-0.6833
4	25	57.2	-32.2	-3.567	53.63	-28.63
5	35	19.25	15.75	-3.567	15.68	+19.32
6	49	57.2	-8.2	7.133	64.33	-15.33
7	68	57.2	10.8	-3.567	53.63	+14.37
8	71	57.2	13.8	7.133	64.33	+6.667
9	73	57.2	15.8	7.133	64.33	+8.667

Gradient Boosting

General Strategy

- Build tree 1, F_1
- Fit a model to residuals, $h_1(x) = y - F_1(x)$
- Create a new model $F_2(x) = F_1(x) + h_1(x)$
- Fit a model to residuals, $h_2(x) = y - F_2(x)$
- Create a new model $F_3(x) = F_2(x) + h_2(x)$
- ...

{13,14,15,25,35,49,68,71,73}

Gardening=F   Gardening=T

{13,14,15,35}   {25,49,68,71,73}

Tree 1

{-6.25,-5.25,-4.25,-32.2,15.75,-8.2,10.8,13.8,15.8}

VideoGames=F   VideoGames=T

{-8.2,13.8,15.8}   {-6.25,-5.25,-4.25,-32.2,15.75,10.8}

Tree 2
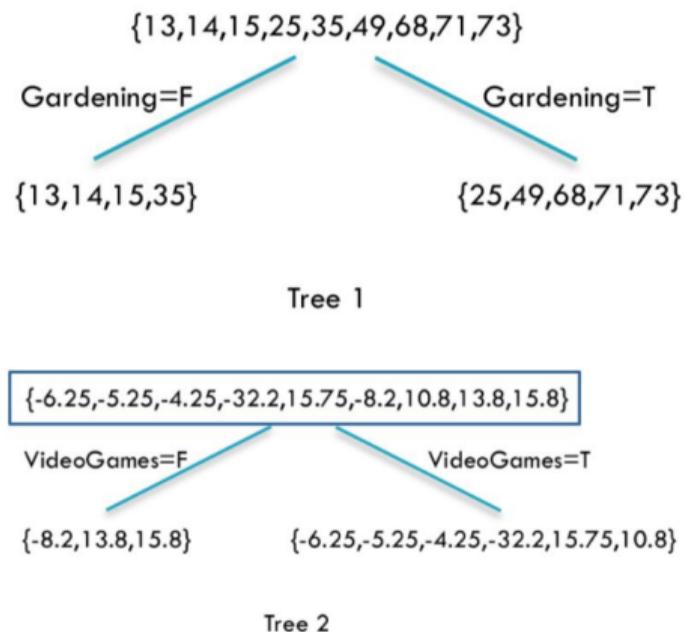
Hyper Parameters

Learning Rate

- h_j fits residuals of F_j
- $F_{j+1}(x) = F_J(x) + LR \cdot h_j(x)$
    - LR controls contribution of residual
    - $LR = 1$ in our previous example
- Ideally, choose LR separately for each residual to minimize loss function
    - Can apply different LR to different leaves

{13,14,15,25,35,49,68,71,73}

Gardening=F   Gardening=T

{13,14,15,35}   {25,49,68,71,73}

Tree 1

{-6.25,-5.25,-4.25,-32.2,15.75,-8.2,10.8,13.8,15.8}

VideoGames=F   VideoGames=T

{-8.2,13.8,15.8}   {-6.25,-5.25,-4.25,-32.2,15.75,10.8}

Tree 2

Residuals and gradients

- Residuals are a special case — gradients for square loss

- Can use other loss functions, and fit h to corresponding gradient

- Square loss gets skewed by outliers

- More robust loss functions with outliers
  - Absolute loss $|y - f(x)|$
  - Huber loss
    $$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2, & |y - F| \leq \delta \\ \delta \cdot (|y - F| - \delta/2), & |y - F| > \delta \end{cases}$$

- More generally, boosting with respect to gradient rather than just residuals

- Given any differentiable loss function L,
  - Start with an initial model F
  - Calculate negative gradients
    $$-g(x_i) = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$
  - Fit a regression tree h to negative gradients $-g(x_i)$
  - Update F to $F + \rho h$
  - ρ is the learning rate

Gradient Boosting for Classification

- Assume binary classification

- Original training outputs are $y \in \{0, 1\}$

- For each x, classifier produces scores $\langle s_0, s_1 \rangle$

- Use softmax to convert to probabilities:

  For $j \in \{0, 1\}$, $p_j = \dfrac{e^{s_j}}{e^{s_0} + e^{s_1}}$

- Use cross entropy as the loss function

  $L(y, F) = y \log(p_1) + (1 - y) \log(p_0)$

- Compute negative gradients

- Fit regression trees to negative gradients to minimize cross entropy