Lecture 12: 17 May, 2021

Madhavan Mukund

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

Data Mining and Machine Learning
April–July 2021

Limitations of classification models

Recall

- Bias : Expressiveness of model limits classification
- Variance: Variation in model based on sample of training data

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?

$D \rightarrow D_1, D_2, \ldots, D_k$

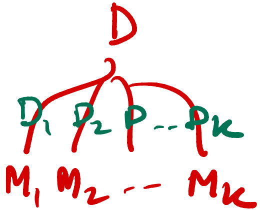
$M_1 \quad M_2 \quad M_k$

Dealing with bias

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

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 $M_1, M_2, \ldots, M_n$ on inputs $D_1, D_2, \ldots, D_n$
  - A weak classifier is any classifier that has error rate strictly below 50%

Boosting

- Build a sequence of weak classifiers $M_1, M_2, \ldots, M_n$ on inputs $D_1, D_2, \ldots, D_n$
  - 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 $\frac{1}{N}$ if $|D| = N$
  - 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

$d_j \in D$

w_j - weight

$\sum w_j = 1$

Decision Stump

Single node decision tree

Boosting

- Build a sequence of weak classifiers $M_1, M_2, \ldots, M_n$ on inputs $D_1, D_2, \ldots, D_n$
  - 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

$\frac{1}{N}$ — β_1 — β_2

$\frac{1}{N}$ $\beta_1 \frac{1}{N}$ $\beta_2 \frac{1}{N}$ $\beta_1 \beta_2 \frac{1}{N}$

D_i M_i

w_1 x_1 ✗ x_{17} ✓ $w_{17} \to w_{17}$

$\beta > 1$, $\beta \cdot w_1$

$D_i : D + W_i$

$M_1 \; M_2 \; - - \; M_k$ — voted

$\alpha_1 \; \alpha_2 \; \ldots \; \alpha_k$

Each model gets an equal vote

Weighted vote

Boosting

- Build a sequence of weak classifiers $M_1, M_2, \ldots, M_n$ on inputs $D_1, D_2, \ldots, D_n$
  - 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

$-(\alpha_{i_1} + \alpha_{i_2} - + \alpha_{i_k})$

$\sum \alpha_i$

$M_i(x) = \cancel{0}\ -1$

$+1, -1$

$\sum \alpha_i$

$M_i(x) = 1$

- Initially, all data items have equal weight

AdaBoost(D, Y, BaseLeaner, k)

$\boxed{\text{Initialize } D_1(w_i) \leftarrow 1/n \text{ for all } i;}$

for $t = 1$ to k **do**

  $f_t \leftarrow \text{BaseLearner}(D_t)$;

  $e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$; $e_t = 0.1$ $\beta_t = \frac{0.1}{0.9}$

  if $e_t > ½$ **then** $e_t \leq \frac{1}{2} = 0.4$

    $k \leftarrow k - 1$;

    exit-loop

  else $\beta_t = \frac{\leq ½}{1-(\leq ½)}$ $\frac{0.4}{0.6}$

    $\beta_t \leftarrow e_t / (1 - e_t)$;

    $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;

    $D_{t+1}(w_i) \leftarrow \frac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$ Normalize

The boosting algorithm — Adaboost

- Initially, all data items have equal weight
- Build a new model and compute its weighted error

AdaBoost(D, Y, BaseLeaner, k)

Initialize $D_1(w_i) \leftarrow 1/n$ for all i;

for $t = 1$ to k **do**

$f_t \leftarrow \text{BaseLearner}(D_t)$;

$e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;

if $e_t > ½$ **then**

$k \leftarrow k - 1$;

exit-loop

else

$\beta_t \leftarrow e_t / (1 - e_t)$;

$$D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases};$$

$$D_{t+1}(w_i) \leftarrow \frac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$$

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%

AdaBoost(D, Y, BaseLeaner, k)

Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
for $t = 1$ to k **do**
    $f_t \leftarrow$ BaseLearner(D_t);
    $e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
    if $e_t > ½$ **then**
        $k \leftarrow k - 1$;
        exit-loop
    else
        $\beta_t \leftarrow e_t / (1 - e_t)$;
        $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
        $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

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)

Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
for $t = 1$ to k **do**
  $f_t \leftarrow$ BaseLearner(D_t);
  $e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
  if $e_t > ½$ **then**
    $k \leftarrow k - 1$;
    exit-loop
  else
    $\boxed{\beta_t \leftarrow e_t / (1 - e_t);}$
    $D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases}$;
    $D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

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)

Initialize $D_1(w_i) \leftarrow 1/n$ for all i;
for $t = 1$ to k **do**
    $f_t \leftarrow$ BaseLearner(D_t);
    $e_t \leftarrow \sum_{i: f_t(D_t(\mathbf{x}_i)) \neq y_i} D_t(w_i)$;
    if $e_t > ½$ **then**
        $k \leftarrow k - 1$;
        exit-loop
    else
        $\beta_t \leftarrow e_t / (1 - e_t)$;

$$D_{t+1}(w_i) \leftarrow D_t(w_i) \times \begin{cases} \beta_t & \text{if } f_t(D_t(\mathbf{x}_i)) = y_i \\ 1 & \text{otherwise} \end{cases};$$

$$D_{t+1}(w_i) \leftarrow \frac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$$

The boosting algorithm — Adaboost

- Initially, all data items have equal weight
- Build a new model and compute its weighted error

  $\alpha_t \sim \cancel{e_t}\ \frac{1}{\beta_t}$

- Discard if error rate is above 50%
- Damping factor — reduce weight of correct inputs
- Reweight data items and normalize
- Final classifier

  $\alpha_t = \log \frac{1}{\beta_t}$

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

$y \in \{0,1\}$

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** $e_t > ½$ **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. $\quad\quad D_{t+1}(w_i) \leftarrow \dfrac{D_{t+1}(w_i)}{\sum_{i=1}^{n} D_{t+1}(w_i)}$

e_t fract. of error

e_t small - good

The boosting algorithm — Adaboost

- Each M_i could be a different type of model

The boosting algorithm — Adaboost

- Each M_i could be a different type of model
- Can we pick best n out of N weak classifiers?

The boosting algorithm — Adaboost

- Each M_i could be a different type of model
- Can we pick best n out of N weak classifiers?
- Initially all data items have equal weight, select M_1 as model with lowest error rate among N candidates

The boosting algorithm — Adaboost

- Each M_i could be a different type of model
- Can we pick best n out of 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 $\alpha_1, \ldots,$ α_j, and dataset is updated with new weights as D_{j+1}

The boosting algorithm — Adaboost

- Each M_i could be a different type of model
- Can we pick best n out of 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 $\alpha_1, \ldots,$ α_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}

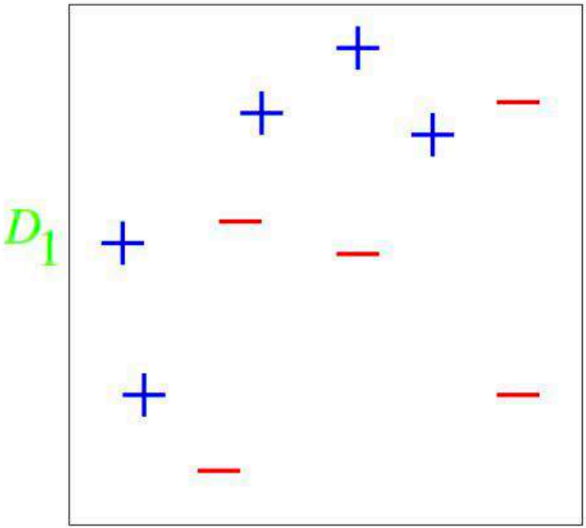
Multiplicative updates

The boosting algorithm — Adaboost

- Each M_i could be a different type of model
- Can we pick best n out of 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 $\alpha_1, \ldots,$ α_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

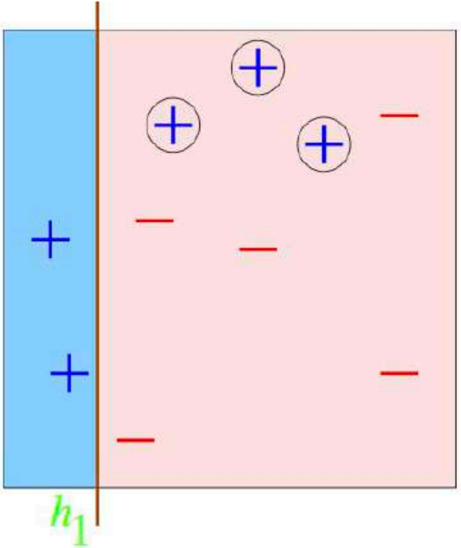
Boosting: An example

- Weak classifiers are horizontal and vertical lines
- Initial training data has equal weights



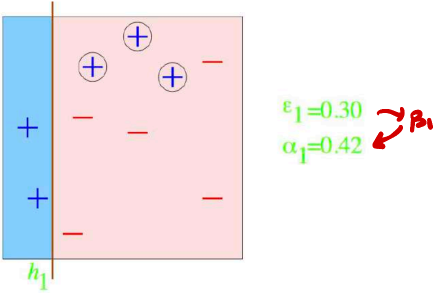
Boosting: An example

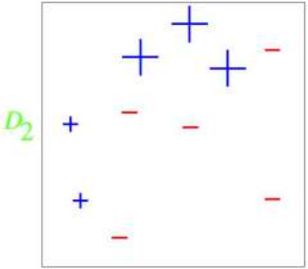
- Weak classifiers are horizontal and vertical lines
- Initial training data has equal weights
- First separator: vertical line



Boosting: An example

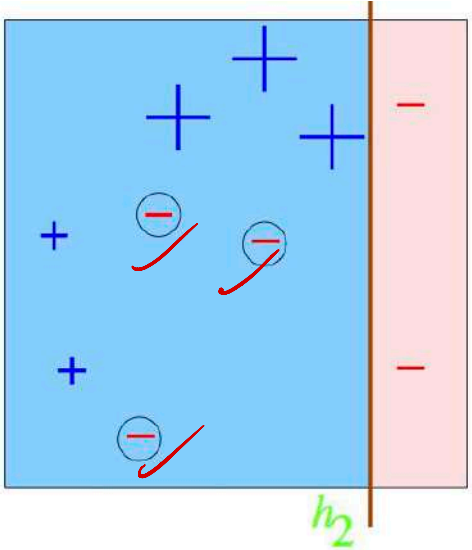
- Weak classifiers are horizontal and vertical lines
- Initial training data has equal weights
- First separator: vertical line
  - Increase weight of misclassified inputs





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

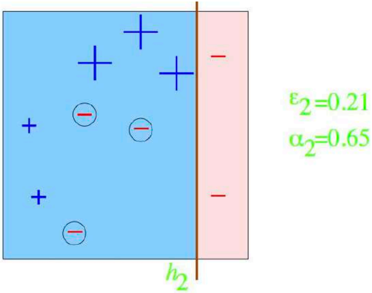


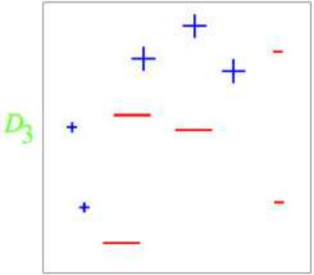
ε_2

α_2

Boosting: An example

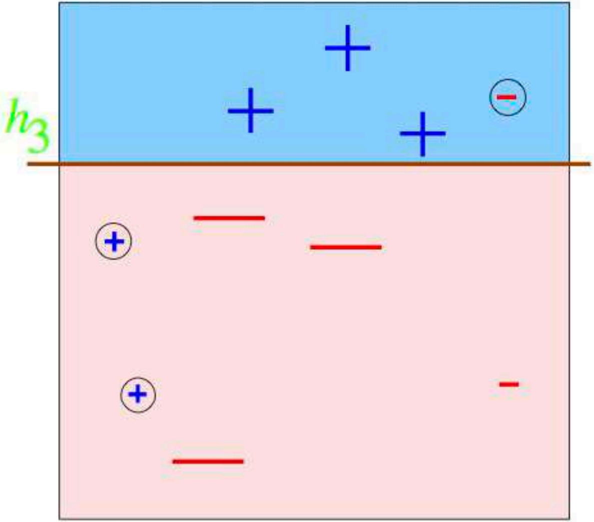
- Weak classifiers are horizontal and vertical lines
- Initial training data has equal weights
- First separator: vertical line
  - Increase weight of misclassified inputs
- Second separator: vertical line
  - Increase weight of misclassified inputs





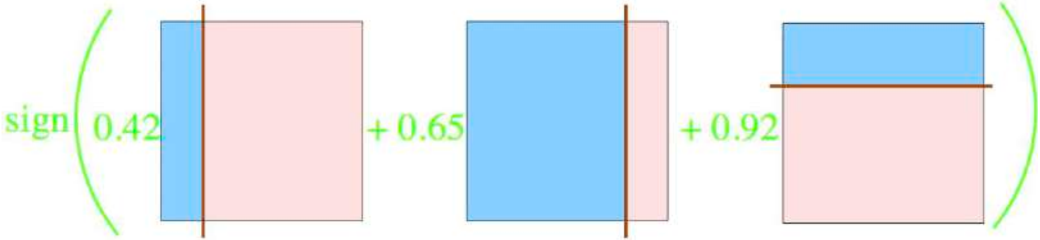
Boosting: An example

- 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



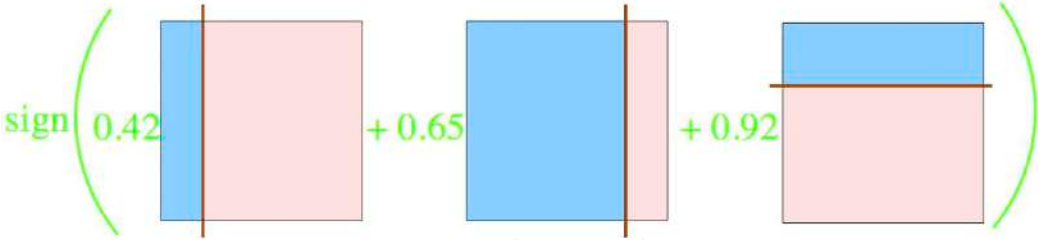
Boosting: An example

- Final classifier is weighted sum of three weak classifiers

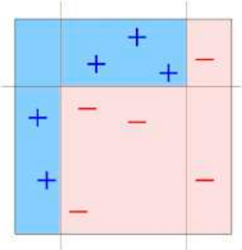


Boosting: An example

- Final classifier is weighted sum of three weak classifiers



- 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

Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots, (x_n, y_n)$
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9,\ F(x_1) = 0.8$
  - $y_2 = 1.3,\ F(x_2) = 1.4$
  - $\ldots$

Gradient Boosting for Regression

- Training data $(x_1, y_1,\ (x_2, y_2), \ldots,$ (x_n, y_n)
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9,\ F(x_1) = 0.8$
  - $y_2 = 1.3,\ F(x_2) = 1.4$
  - $\ldots$
- Add an additional model h, so that new prediction is $F(x) + h(x)$

Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots,$ (x_n, y_n)
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9,\ F(x_1) = 0.8$
  - $y_2 = 1.3,\ F(x_2) = 1.4$
  - $\ldots$
- Add an additional model h, so that new prediction is $F(x) + h(x)$
- What should h look like?

Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots, (x_n, y_n)$
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9,\ F(x_1) = 0.8$
  - $y_2 = 1.3,\ F(x_2) = 1.4$
  - $\ldots$
- Add an additional model h, so that new prediction is $F(x) + h(x)$

- What should h look like?
- For each x_i, want $F(x_i) + h(x_i) = y_i$

Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots, (x_n, y_n)$
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9$, $F(x_1) = 0.8$
  - $y_2 = 1.3$, $F(x_2) = 1.4$
  - $\ldots$
- Add an additional model h, so that new prediction is $F(x) + h(x)$
- What should h look like?
- For each x_i, want $F(x_i) + h(x_i) = y_i$
- $h(x_i) = y_i - F(x_i)$

Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots, (x_n, y_n)$
- 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 $y_i - F(x_i)$

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 $y_i - [F(x_i) + h(x_i)]$

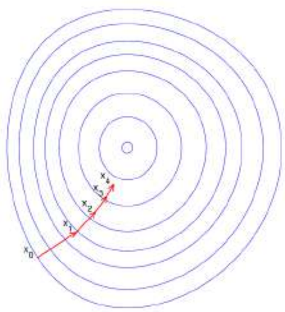
Gradient Boosting for Regression

- Training data $(x_1, y_1, (x_2, y_2), \ldots, (x_n, y_n)$
- Fit a model $F(x)$ to minimize square loss
- The model F we build is good, but not perfect
  - $y_1 = 0.9$, $F(x_1) = 0.8$
  - $y_2 = 1.3$, $F(x_2) = 1.4$
  - $\ldots$
- 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?

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i} \cdot \delta$$

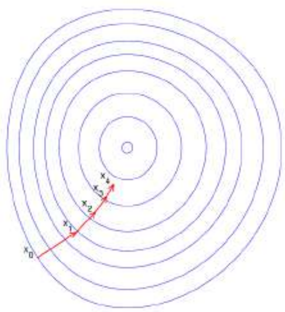


Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i}$$



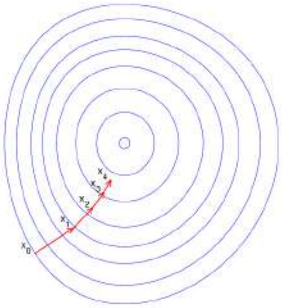
- Individual loss:
  $L(y, F(x)) = (y - F(x))^2/2$

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:
  $J = \sum_i L(y_i, F(x_i))$

Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

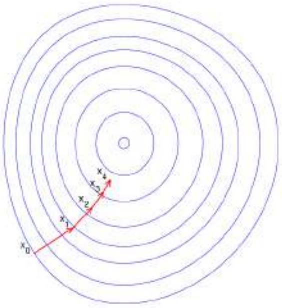
$\theta_0+\theta_1 x_1+\theta_2 x_2$

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i}$$

Loss fn

$\uparrow$

$\frac{\partial Loss}{\partial \theta_i}$



- Individual loss:
  $L(y, F(x)) = (y - F(x))^2/2$
- Minimize overall loss:
  $J = \sum_i L(y_i, F(x_i))$
- $\frac{\partial J}{\partial F(x_i)} = F(x_i) - y_i$

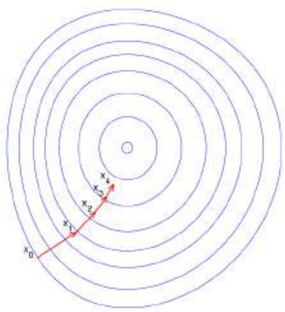
$\frac{2 \cdot (y_i - F(x_i))}{2} \cdot -1$

Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i}$$



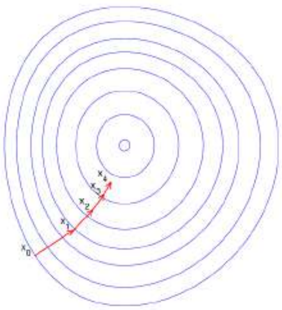
- Individual loss:
  $L(y, F(x)) = (y - F(x))^2/2$
- Minimize overall loss:
  $J = \sum_i L(y_i, F(x_i))$
- $\dfrac{\partial J}{\partial F(x_i)} = F(x_i) - y$
- Residual $y_i - F(x_i)$ is negative gradient

Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i}$$



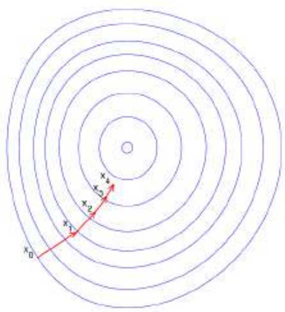
- Individual loss: $L(y, F(x)) = (y - F(x))^2/2$
- Minimize overall loss: $J = \sum_i L(y_i, F(x_i))$
- $\frac{\partial J}{\partial F(x_i)} = F(x_i) - y$
- Residual $y_i - F(x_i)$ is negative gradient
- Fitting h to residual is same as fitting h to negative gradient

Residuals and gradients

Gradient descent

- Move parameters against the gradient with respect to loss function

$$\theta_i \leftarrow \theta_i - \frac{\partial J}{\partial \theta_i} \quad \delta$$



- Individual loss:
  $L(y, F(x)) = (y - F(x))^2/2$
- Minimize overall loss:
  $J = \sum_i L(y_i, F(x_i))$

$F + \delta h$

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

Residuals and gradients

- Residuals are a special case — gradients for square loss

Residuals and gradients

- Residuals are a special case — gradients for square loss
- Can use other loss functions, and fit h to corresponding gradient

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

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

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2, & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2), & |y - F| > \delta \end{cases}$$

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

$$L(y, F) = \begin{cases} \frac{1}{2}(y-F)^2, & |y-F| \leq \delta \\ \delta(|y-F| - \delta/2), & |y-F| > \delta \end{cases}$$

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

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

$$L(y, F) = \begin{cases} \frac{1}{2}(y-F)^2, & |y-F| \leq \delta \\ \delta(|y-F| - \delta/2), & |y-F| > \delta \end{cases}$$

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

$y_i - F(x_i) = V_i$

$x_1 - y_1$
$x_2 \; y_2$
$x_n - y_n$ F

$x_1 \to V_1$
$x_2 \to V_2$
$\vdots$
$x_n \to V_n$