Gradient Boosting

Evgeny Burnaev

Skoltech, Moscow, Russia

Outline

Classification Problem

- A predictor, feature $\mathbf{x} \in \mathbb{R}^d$ has distribution D
- $f(\mathbf{x})$ is a deterministic function from some concept class
- **Goal**:
  - Based on m training pairs $\{(\mathbf{x}_i, y_i = f(\mathbf{x}_i))\}_{i=1}^m$ drawn i.i.d. from D produce a classifier $\widehat{f}(\mathbf{x}) \in \{0, 1\}$
  - Choose $\widehat{f}$ to have low generalization error $R(\widehat{f}) = \mathbb{E}_D \left[1_{\widehat{f}(\mathbf{x}) \neq y}\right]$

Classification Problem

- A predictor, feature $\mathbf{x} \in \mathbb{R}^d$ has distribution D
- $f(\mathbf{x})$ is a deterministic function from some concept class
- **Goal**:
  - Based on m training pairs $\{(\mathbf{x}_i, y_i = f(\mathbf{x}_i))\}_{i=1}^m$ drawn i.i.d. from D produce a classifier $\widehat{f}(\mathbf{x}) \in \{0, 1\}$
  - Choose $\widehat{f}$ to have low generalization error $R(\widehat{f}) = \mathbb{E}_D \left[1_{\widehat{f}(\mathbf{x}) \neq y}\right]$

Ensemble of classifiers

- For simplicity we consider a binary classification problem. Let us denote by $h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})$ some binary classifiers
- Typical ensembling procedure has the form
  - Simple voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x}),$$

  - Weighted voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}),$$

  - Mixture of experts

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} g_t(\mathbf{x}) h_t(\mathbf{x})$$

- Final decision

$$f_T(\mathbf{x}) = \text{sign}\{H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x}))\}$$

Ensemble of classifiers

- For simplicity we consider a binary classification problem. Let us denote by $h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})$ some binary classifiers
- Typical ensembling procedure has the form
  - Simple voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x}),$$

  - Weighted voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}),$$

  - Mixture of experts

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} g_t(\mathbf{x}) h_t(\mathbf{x})$$

- Final decision

$$f_T(\mathbf{x}) = \text{sign}\{H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x}))\}$$

Ensemble of classifiers

- For simplicity we consider a binary classification problem. Let us denote by $h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})$ some binary classifiers
- Typical ensembling procedure has the form
  - Simple voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x}),$$

  - Weighted voting:

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}),$$

  - Mixture of experts

$$H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})) = \sum_{t=1}^{T} g_t(\mathbf{x}) h_t(\mathbf{x})$$

- Final decision

$$f_T(\mathbf{x}) = \operatorname{sign}\{H(h_1(\mathbf{x}), \ldots, h_T(\mathbf{x}))\}$$

Boosting for regression

- Consider a regression problem $\frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - y_i)^2 \to \min_h$
- Search for solution in the form of weak learner composition $f_T(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x})$ with weak learners $h_t \in \mathbb{H}$
- The **boosting approach:** add weak learners greedily
  1. Start with a "trivial" weak learner $h_0(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} y_i$
  2. At step T, compute the *residuals*

$$s_i^{(T)} = y_i - \sum_{t=1}^{T-1} h_t(\mathbf{x}_i) = y_i - f_{T-1}(\mathbf{x}_i), \qquad i = 1, \ldots, m$$

  3. Learn the next weak algorithm using

$$h_T(\mathbf{x}) := \operatorname*{argmin}_{h \in \mathbb{H}} \frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - s_i^{(T)})^2$$

(this implementation may be found in, e.g., `scikit-learn`)

Boosting for regression

- Consider a regression problem $\frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - y_i)^2 \to \min_h$
- Search for solution in the form of weak learner composition $f_T(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x})$ with weak learners $h_t \in \mathbb{H}$
- The **boosting approach:** add weak learners greedily
  1. Start with a "trivial" weak learner $h_0(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} y_i$
  2. At step T, compute the *residuals*

$$s_i^{(T)} = y_i - \sum_{t=1}^{T-1} h_t(\mathbf{x}_i) = y_i - f_{T-1}(\mathbf{x}_i), \qquad i = 1, \ldots, m$$

  3. Learn the next weak algorithm using

$$h_T(\mathbf{x}) := \operatorname*{argmin}_{h \in \mathbb{H}} \frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - s_i^{(T)})^2$$

(this implementation may be found in, e.g., `scikit-learn`)

Boosting for regression

- Consider a regression problem $\frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - y_i)^2 \to \min\limits_{h}$
- Search for solution in the form of weak learner composition $f_T(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x})$ with weak learners $h_t \in \mathbb{H}$
- The **boosting approach:** add weak learners greedily
  1. Start with a "trivial" weak learner $h_0(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} y_i$
  2. At step T, compute the *residuals*

$$s_i^{(T)} = y_i - \sum_{t=1}^{T-1} h_t(\mathbf{x}_i) = y_i - f_{T-1}(\mathbf{x}_i), \qquad i = 1, \ldots, m$$

  3. Learn the next weak algorithm using

$$h_T(\mathbf{x}) := \operatorname*{argmin}_{h \in \mathbb{H}} \frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - s_i^{(T)})^2$$

(this implementation may be found in, e.g., `scikit-learn`)

Boosting for regression

- Consider a regression problem $\frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - y_i)^2 \to \min\limits_{h}$
- Search for solution in the form of weak learner composition $f_T(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x})$ with weak learners $h_t \in \mathbb{H}$
- The **boosting approach:** add weak learners greedily
  1. Start with a "trivial" weak learner $h_0(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} y_i$
  2. At step T, compute the *residuals*

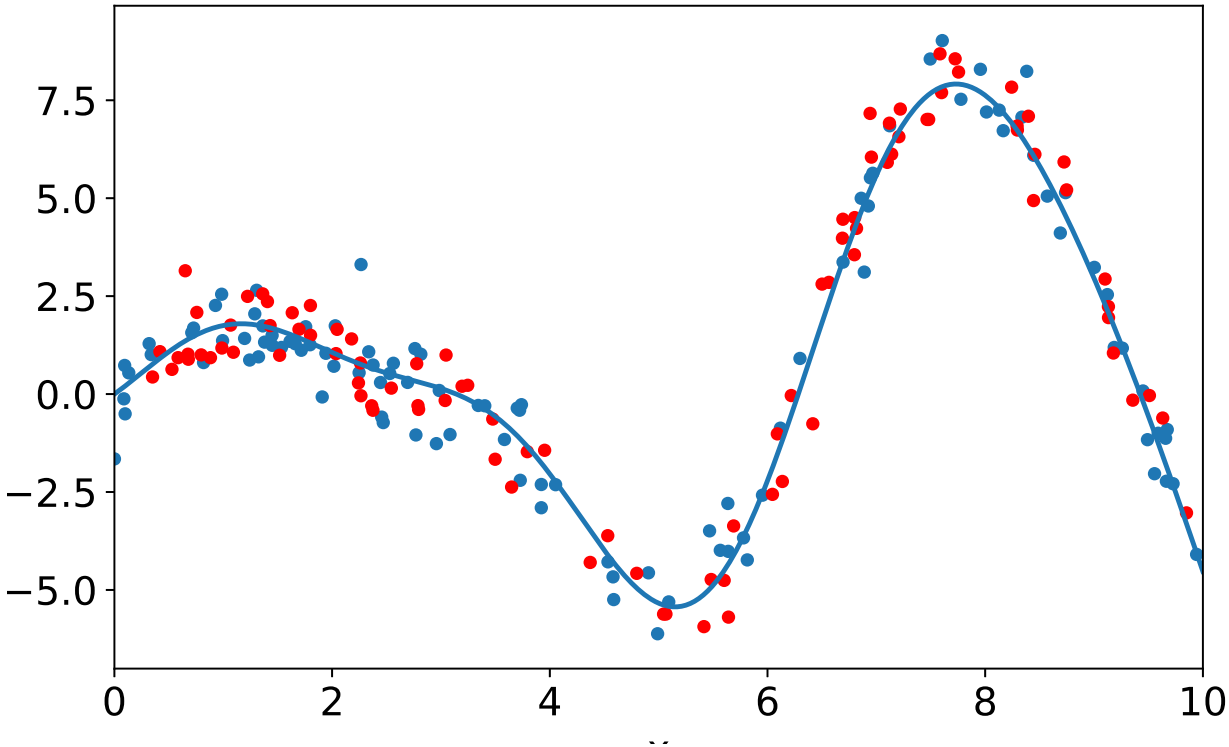
$$s_i^{(T)} = y_i - \sum_{t=1}^{T-1} h_t(\mathbf{x}_i) = y_i - f_{T-1}(\mathbf{x}_i), \qquad i = 1, \ldots, m$$

  3. Learn the next weak algorithm using

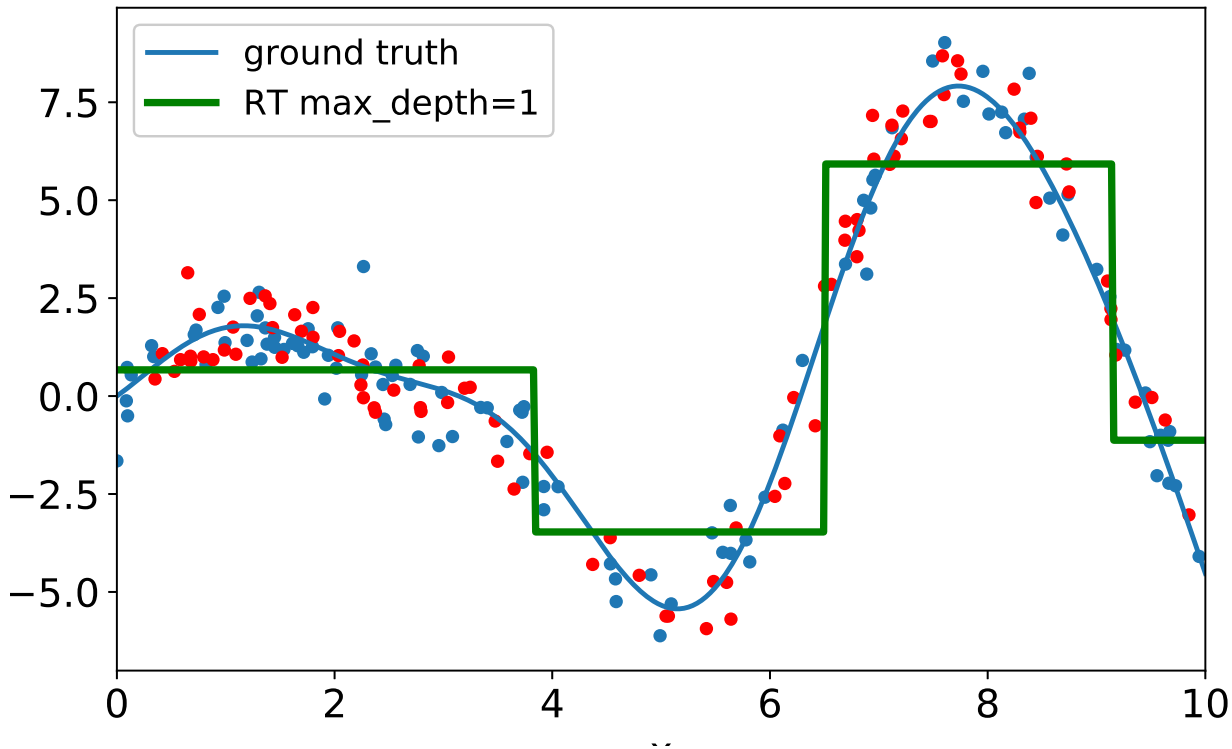
$$h_T(\mathbf{x}) := \operatorname*{argmin}_{h \in \mathbb{H}} \frac{1}{2}\sum_{i=1}^{m}(h(\mathbf{x}_i) - s_i^{(T)})^2$$

(this implementation may be found in, e.g., `scikit-learn`)

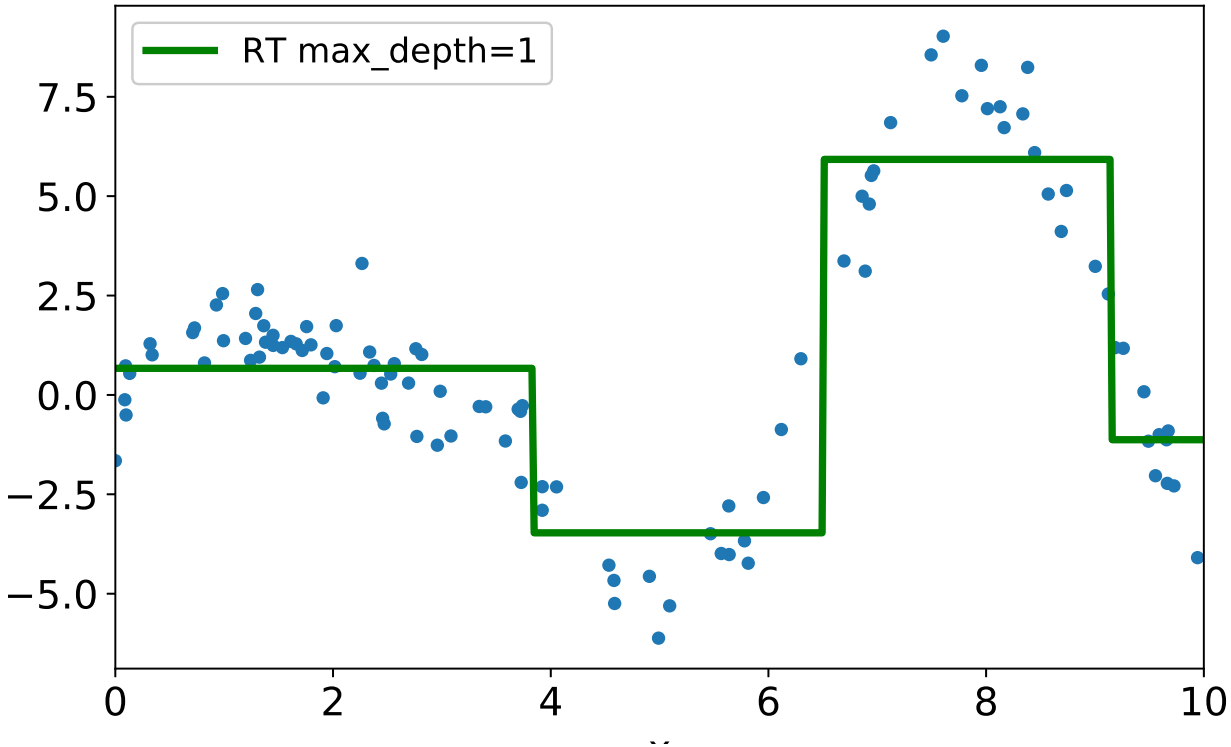
Boosting: an example regression problem



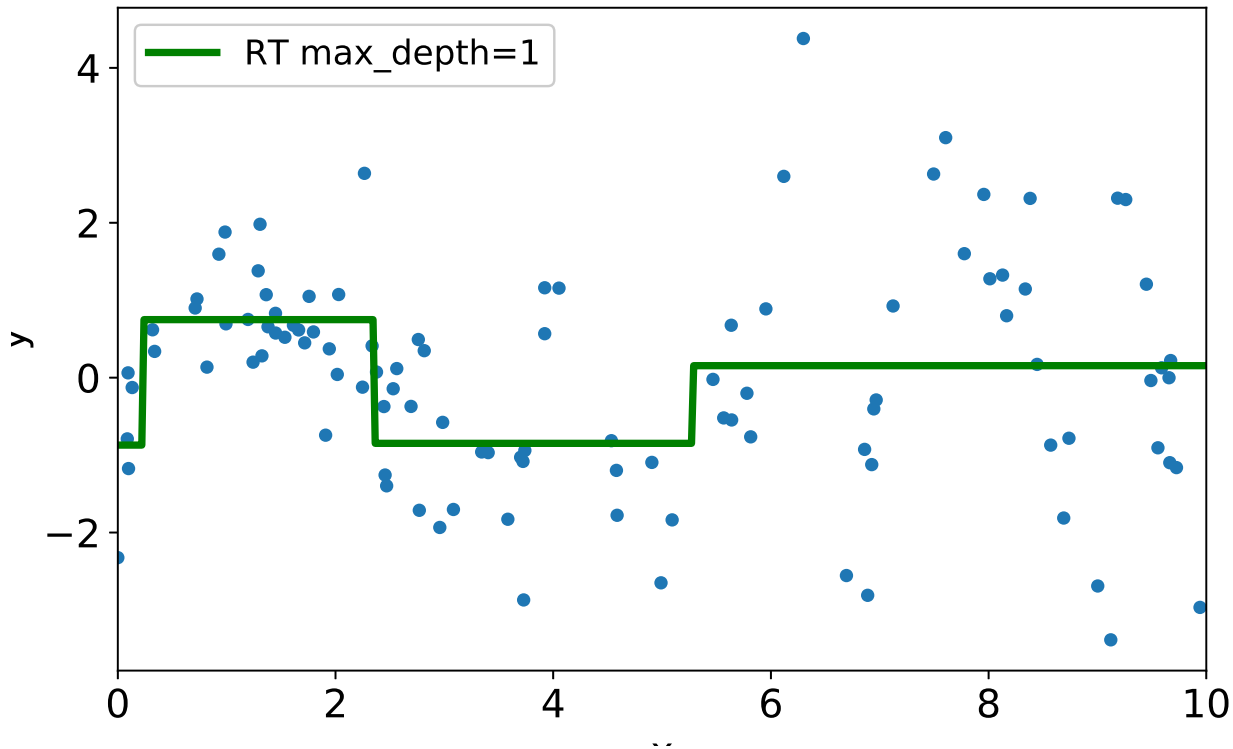
Boosting: an example regression problem



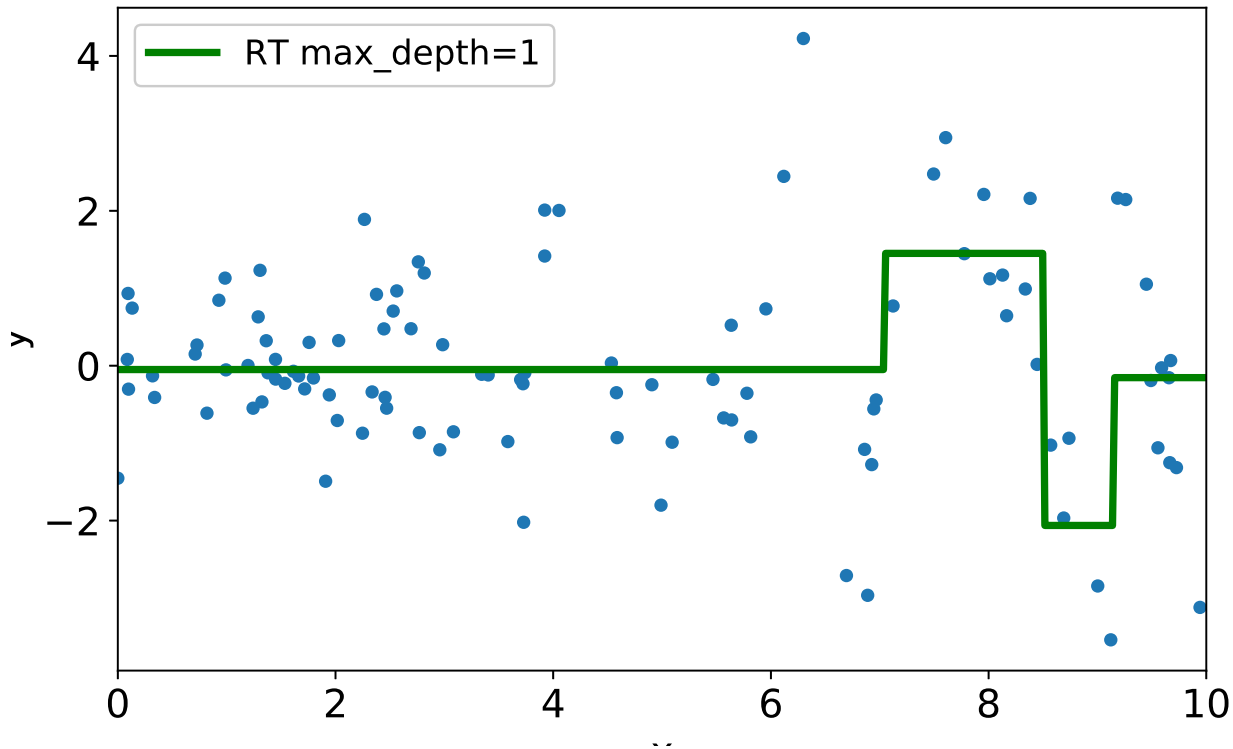
Boosting: an example regression problem



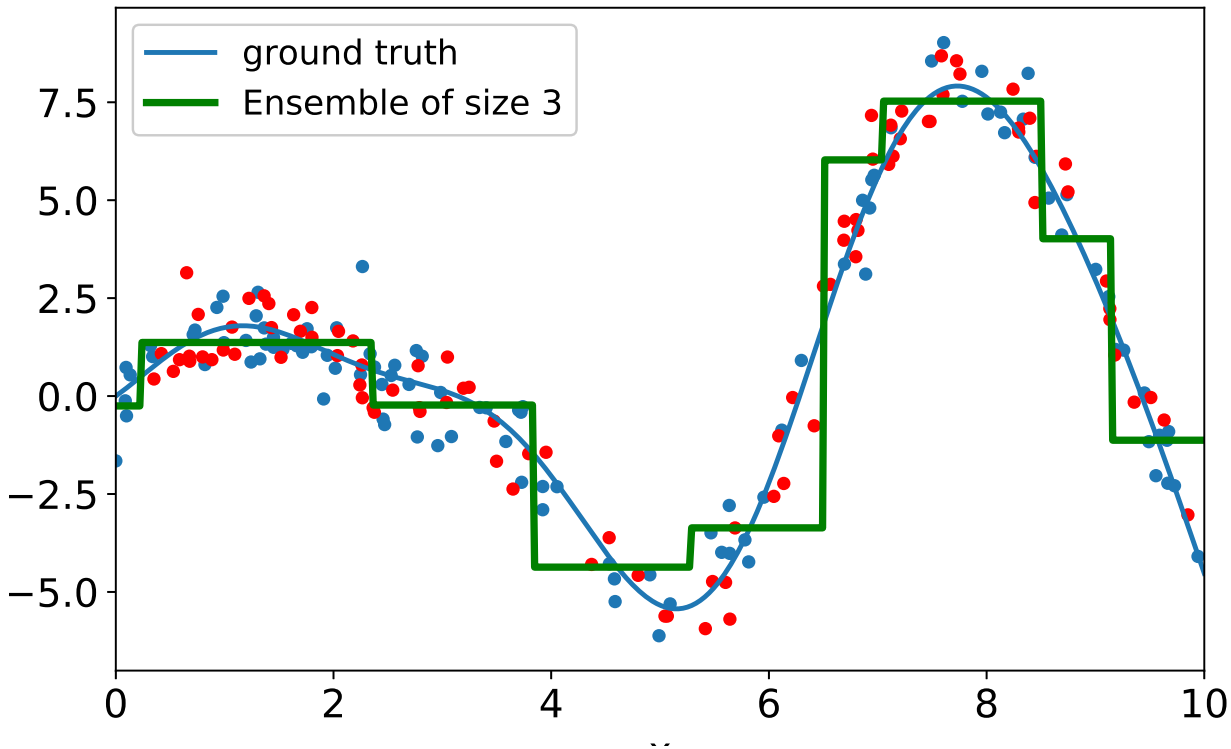
Boosting: an example regression problem



Boosting: an example regression problem



Boosting: an example regression problem



Gradient boosting: motivation

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1,\ldots,s_m\},\alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

Gradient boosting: motivation

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1, \ldots, s_m\}, \alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

Gradient boosting: motivation

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1,\ldots,s_m\},\alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1,\ldots,s_m\},\alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

Gradient boosting: motivation

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1,\ldots,s_m\},\alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i,z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

- With $f_{T-1}(\mathbf{x})$ already built, how to find the next α_T and h_T if

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha_T h_T(\mathbf{x}_i)) \to \min_{\alpha_T, h_T}$$

- Recall: functions decrease in the direction of negative gradient
- View $L(y, z)$ as a function of z $(= f_T(\cdot))$, execute gradient descent on z
- Search for such $s_1, \ldots, s_m$ that

$$\sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha \cdot s_i) \to \min_{\{s_1,\ldots,s_m\},\alpha}$$

- Choose $s_i = -\left.\frac{\partial L(y_i,z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}$
- Approximate s_i's by $h_T(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}_i$ and tune α

The Gradient Boosting Machine [Friedman, 2001]

- Input:
  - Training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
  - Number of boosting iterations T
  - Loss function $L(y, z)$ with its gradient $\frac{\partial L}{\partial z}$
  - A family $\mathbb{H} = \{h(\mathbf{x})\}$ of weak learners and their associated learning procedures
  - Additional hyperparameters of weak learners (tree depth, etc.)
- Initialize GBM $h_0(\mathbf{x})$ using some simple rule (zero, most popular class, etc.)
- Execute boosting iterations $t = 1, \ldots, T$ (see next slide)
- Compose the final GBM learner: $f_T(\mathbf{x}) = \sum_{t=0}^T \alpha_t h_t(\mathbf{x})$

At every iteration:

1. Compute pseudo-residuals: $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}, i = 1, \ldots, m$
2. Learn $h_T(\mathbf{x}_i)$ by regressing onto $s_1, \ldots, s_m$:

$$h_T(\mathbf{x}) = \operatorname*{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{m} \left(h(\mathbf{x}_i) - s_i\right)^2$$

3. Find the optimal α_N using plain gradient descent:

$$\alpha_T = \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha h_T(\mathbf{x}_i))$$

4. Update the GBM by $f_T(\mathbf{x}) \leftarrow f_{T-1}(\mathbf{x}) + \alpha_T h_T(\mathbf{x})$

At every iteration:

1. Compute pseudo-residuals: $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}, i = 1, \ldots, m$
2. Learn $h_T(\mathbf{x}_i)$ by regressing onto $s_1, \ldots, s_m$:

$$h_T(\mathbf{x}) = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(h(\mathbf{x}_i) - s_i\right)^2$$

3. Find the optimal α_N using plain gradient descent:

$$\alpha_T = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha h_T(\mathbf{x}_i))$$

4. Update the GBM by $f_T(\mathbf{x}) \leftarrow f_{T-1}(\mathbf{x}) + \alpha_T h_T(\mathbf{x})$

At every iteration:

1. Compute pseudo-residuals: $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}, i = 1, \ldots, m$
2. Learn $h_T(\mathbf{x}_i)$ by regressing onto $s_1, \ldots, s_m$:

$$h_T(\mathbf{x}) = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(h(\mathbf{x}_i) - s_i\right)^2$$

3. Find the optimal α_N using plain gradient descent:

$$\alpha_T = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha h_T(\mathbf{x}_i))$$

4. Update the GBM by $f_T(\mathbf{x}) \leftarrow f_{T-1}(\mathbf{x}) + \alpha_T h_T(\mathbf{x})$

At every iteration:

1. Compute pseudo-residuals: $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=f_{T-1}(\mathbf{x}_i)}, i = 1, \ldots, m$
2. Learn $h_T(\mathbf{x}_i)$ by regressing onto $s_1, \ldots, s_m$:

$$h_T(\mathbf{x}) = \underset{h \in \mathbb{H}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(h(\mathbf{x}_i) - s_i\right)^2$$

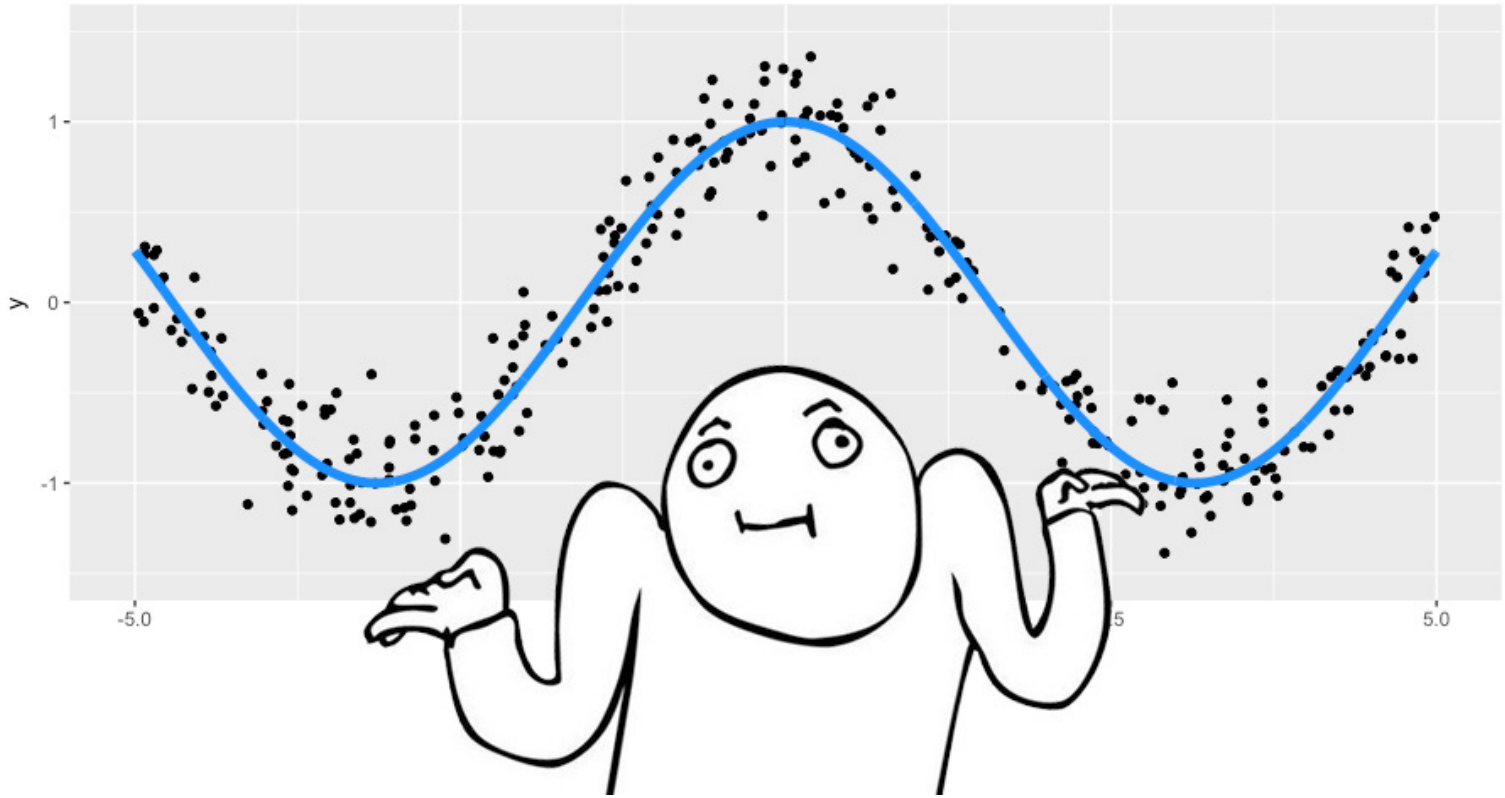
3. Find the optimal α_N using plain gradient descent:

$$\alpha_T = \underset{\alpha \in \mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{m} L(y_i, f_{T-1}(\mathbf{x}_i) + \alpha h_T(\mathbf{x}_i))$$

4. Update the GBM by $f_T(\mathbf{x}) \leftarrow f_{T-1}(\mathbf{x}) + \alpha_T h_T(\mathbf{x})$

GBM: an example regression problem

- Consider a training set for a $S_{300} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{300}$ where $\mathbf{x}_i \in [-5, 5]$, $y_i = \cos(x_i) + \varepsilon_i$, $\varepsilon_i \sim \mathcal{N}(0, 1/5)$

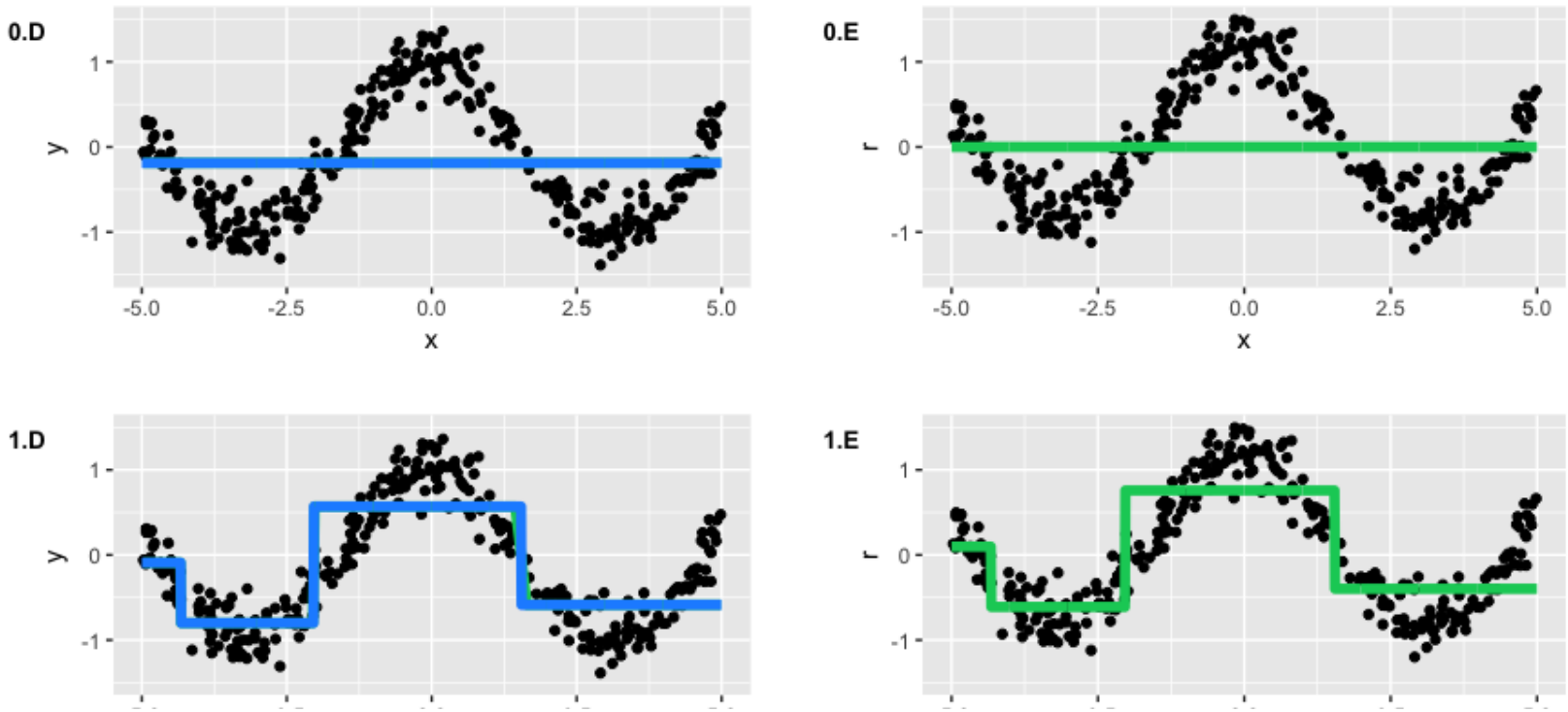


Picture credit: https://habrahabr.ru/company/ods/blog/327250

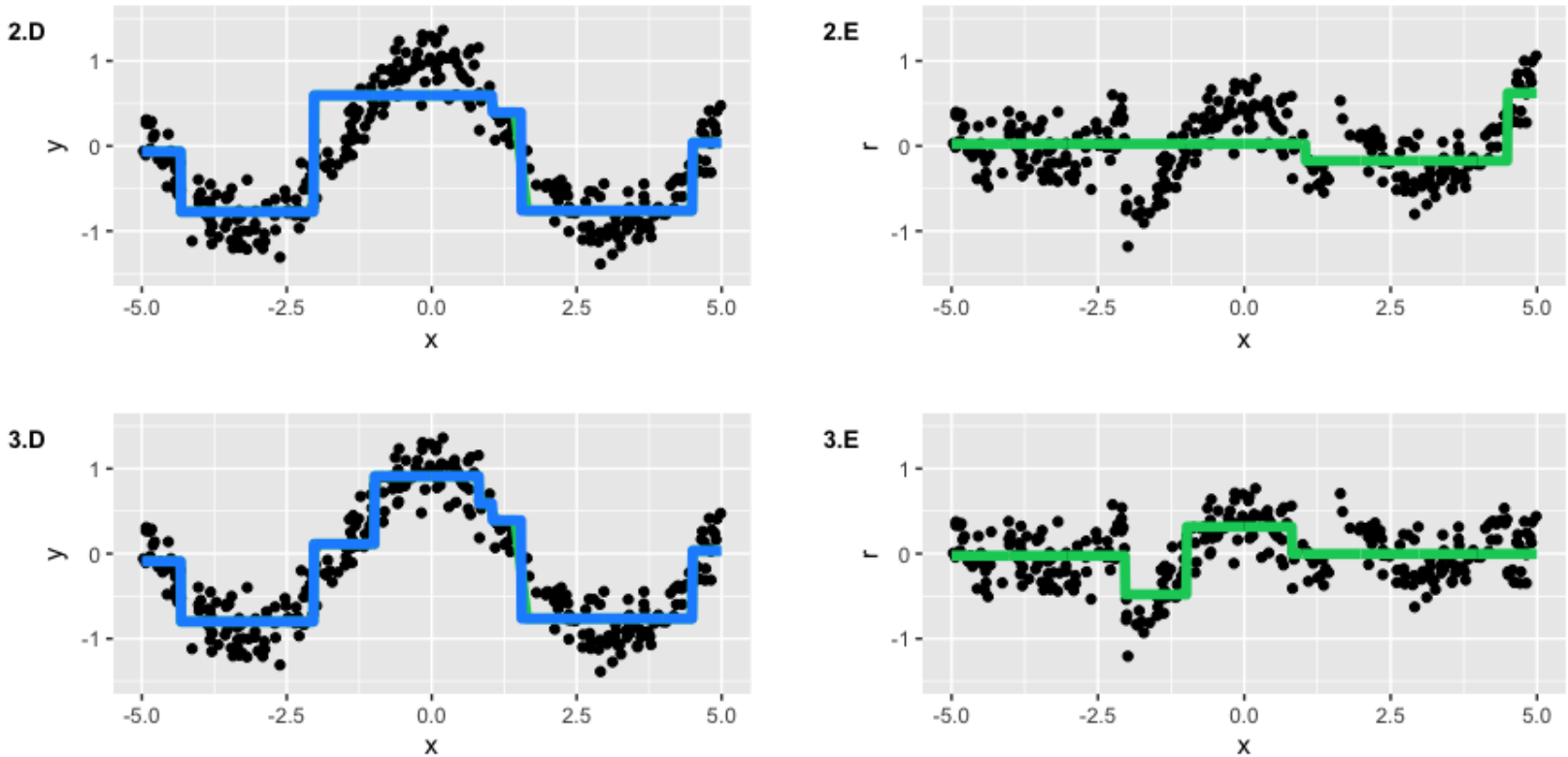
GBM: an example regression problem

- Pick $T = 3$ boosting iterations
- Quadratic loss $L(y, z) = (y - z)^2$
- Gradient of the quadratic loss $\frac{\partial L(y_i, z)}{\partial z} = (y - z)$ is just redisuals
- Pick decision trees as weak learners $h_t(\mathbf{x})$
- Set 2 as the maximum depth for decision trees

GBM: an example regression problem

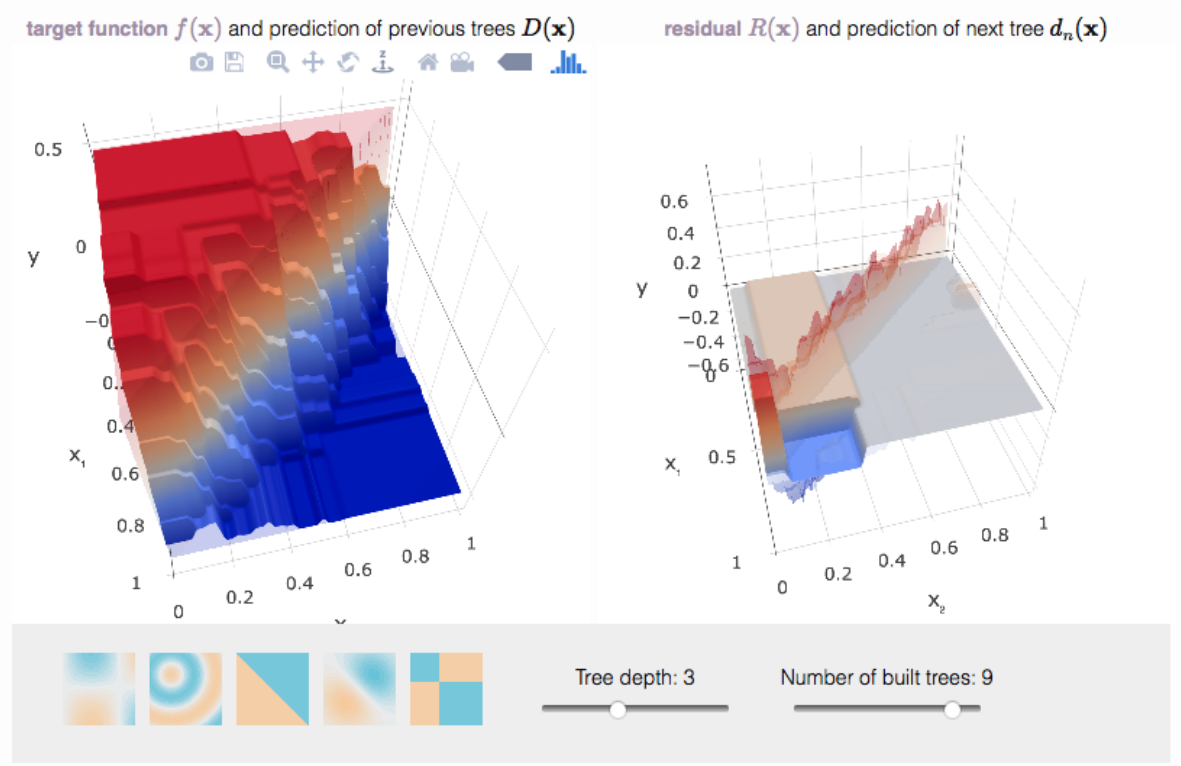


GBM: an example regression problem



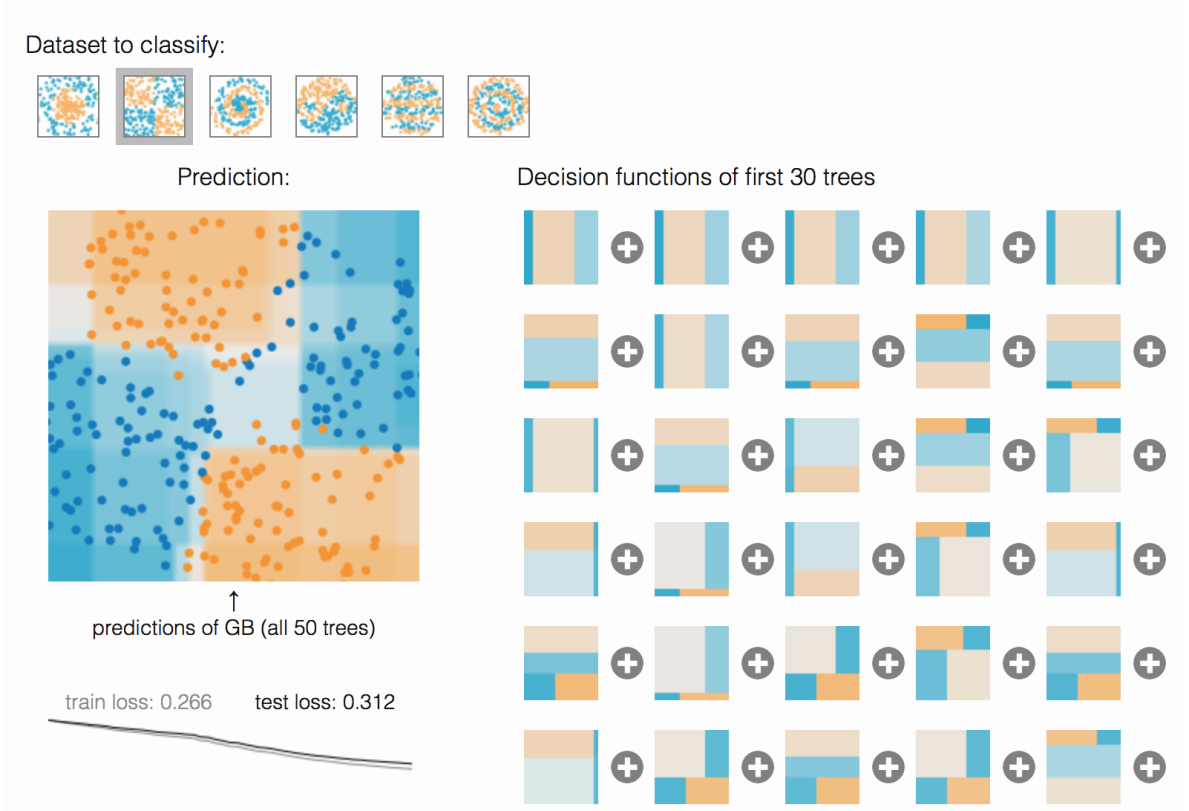
GBM: an interactive demo

http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html



GBM: an interactive demo

http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html



GBM: regulization via shrinkage

- For too simple weak learners, the negative gradient is approximated badly $\Longrightarrow$ random walk in space of samples
- For too complex weak learners, a few boosting steps may be enough for overfitting
- **Shrinkage:** make shorter steps using a *learning rate* $\eta \in (0, 1]$

$$f_T(\mathbf{x}_i) \leftarrow f_{T-1}(\mathbf{x}) + \eta \alpha_T h_T(\mathbf{x})$$

(effectively distrust gradient direction estimated via weak learners)

GBM: shrinkage animated

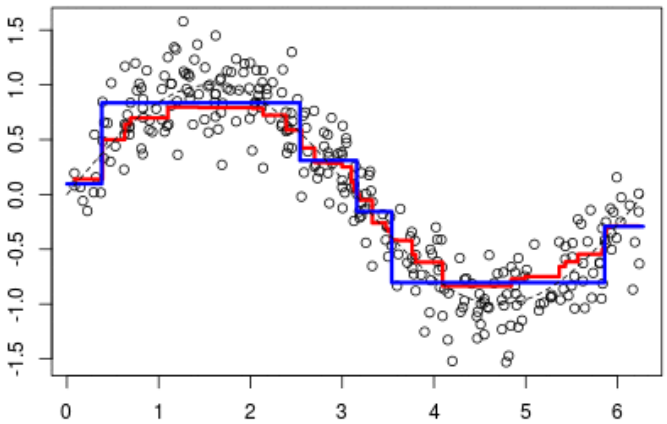


Figure – High shrinkage

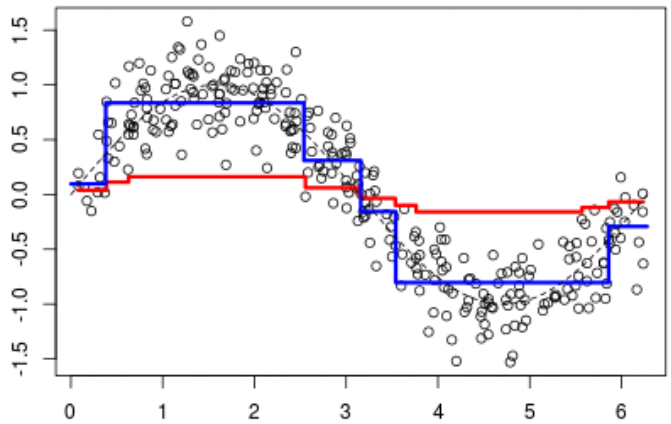
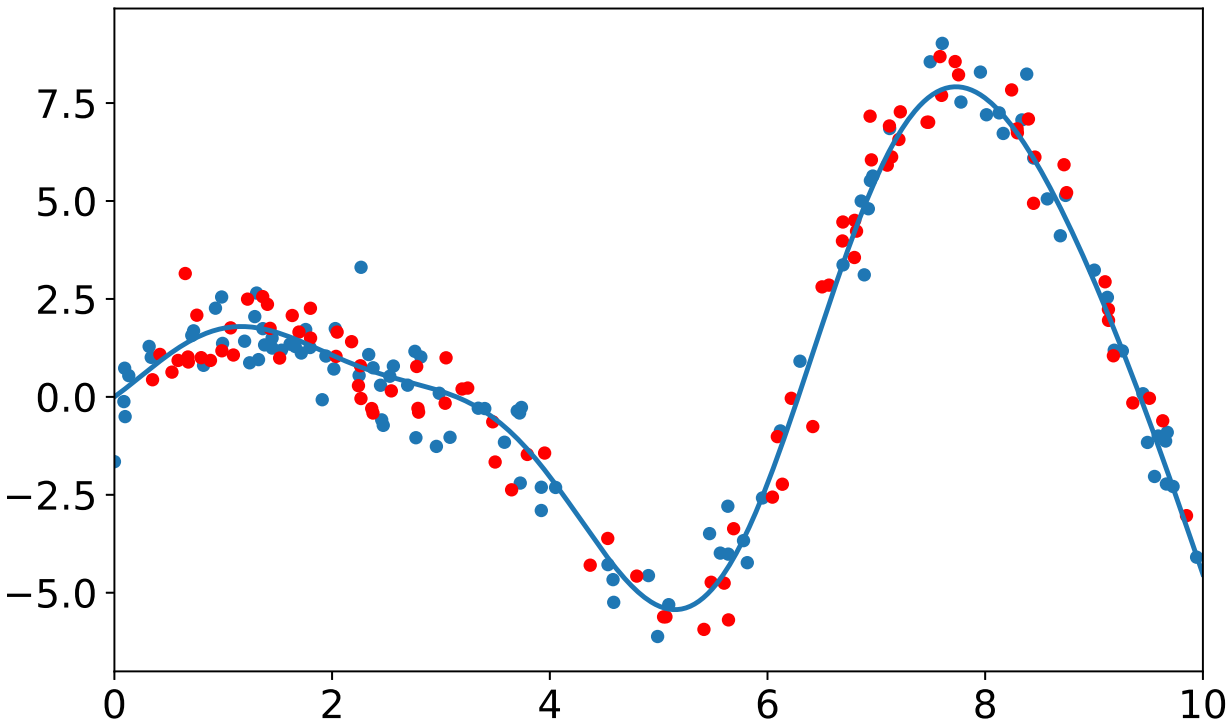
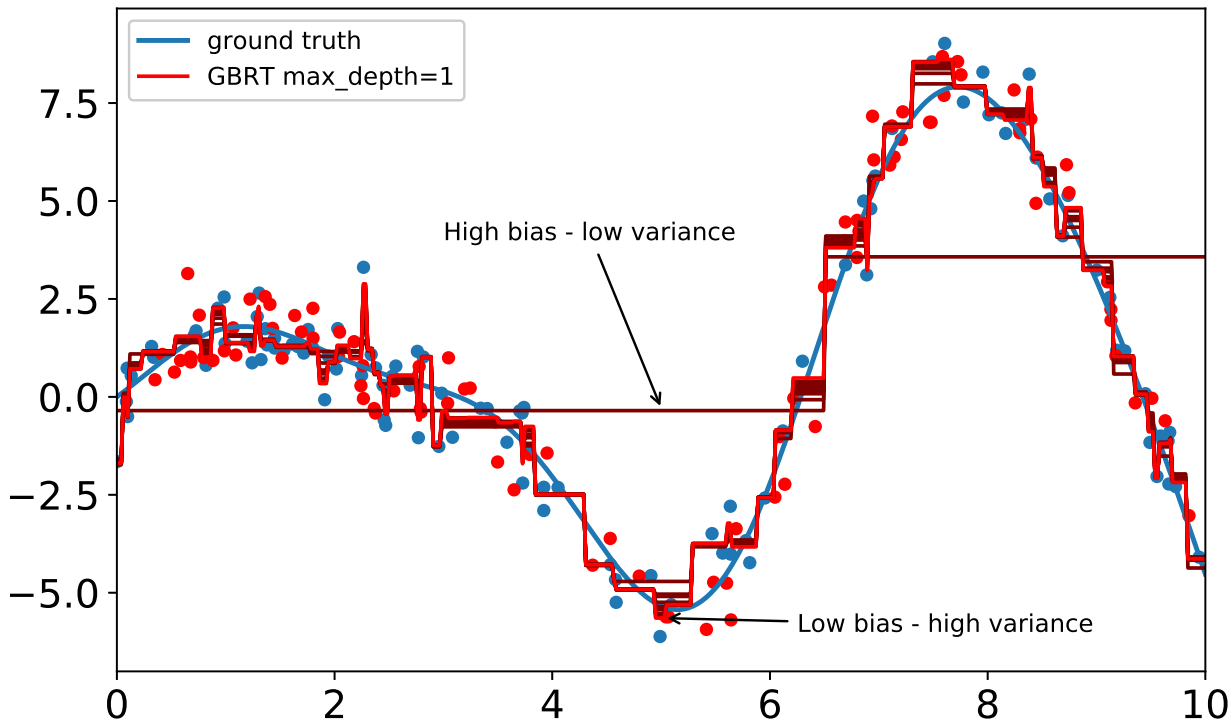


Figure – Low shrinkage

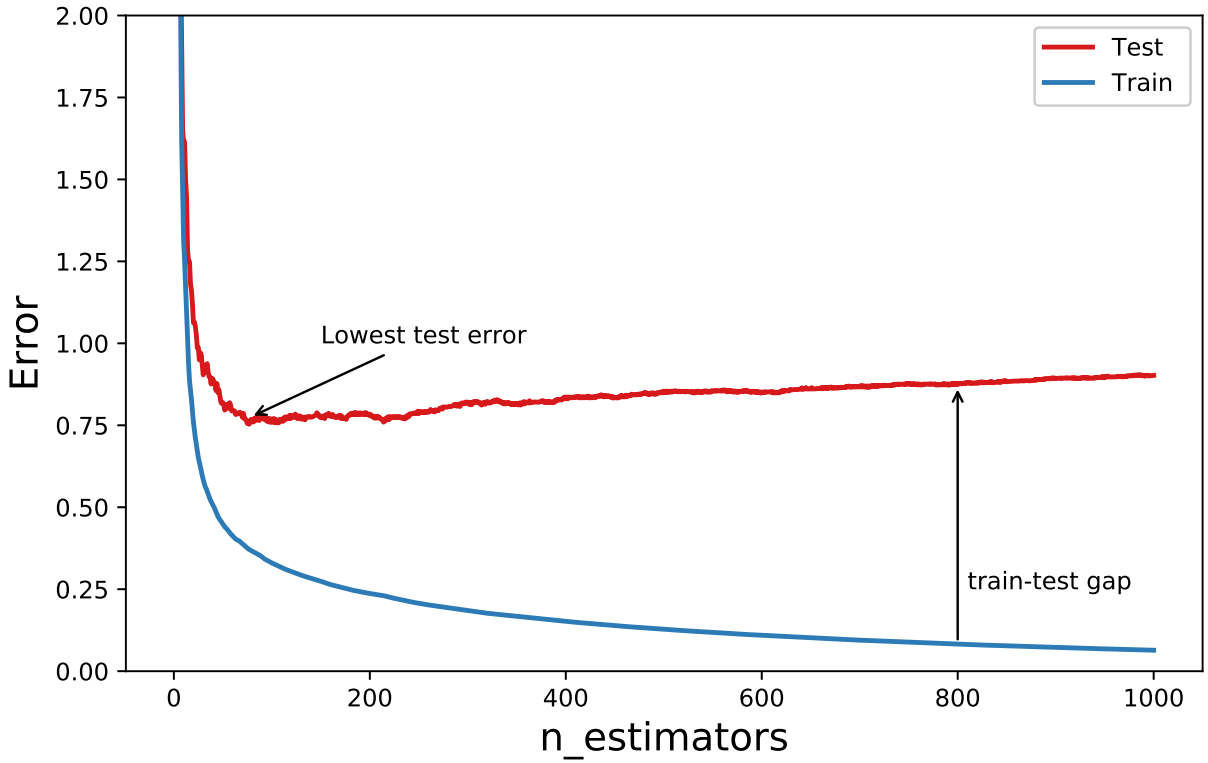
GBM: regularization approaches



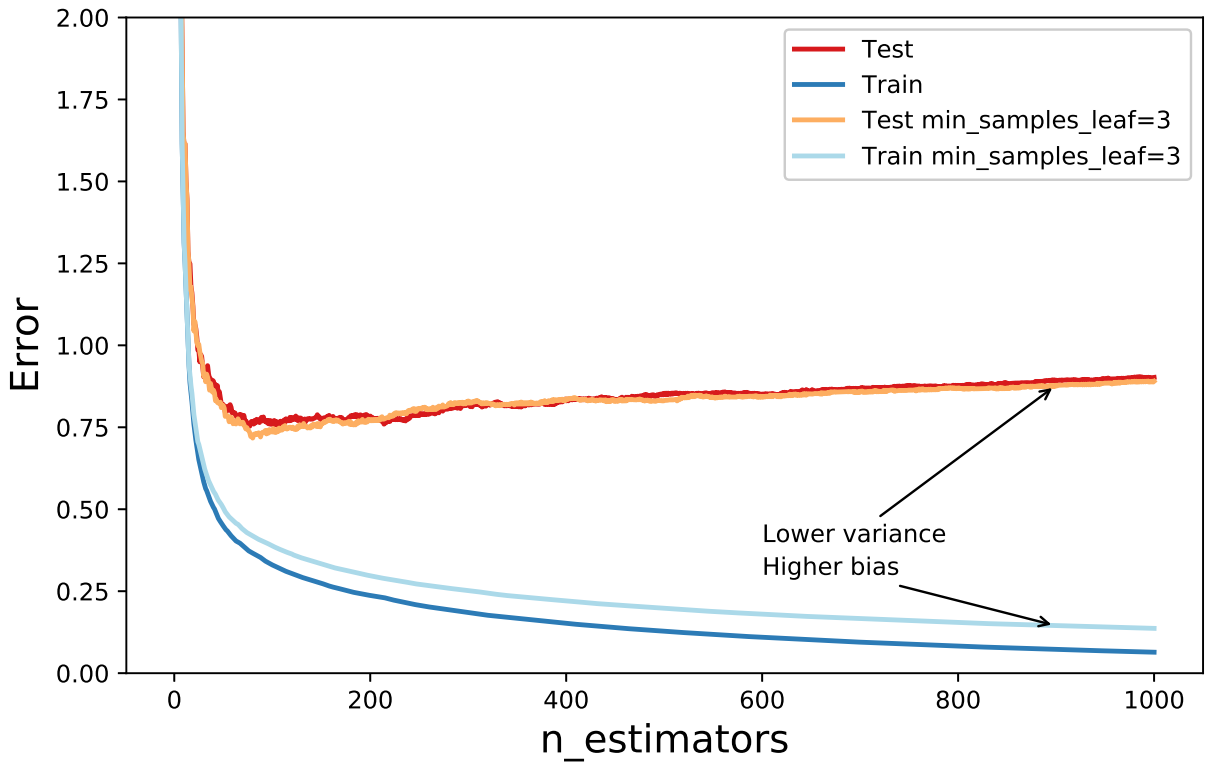
GBM: regularization approaches



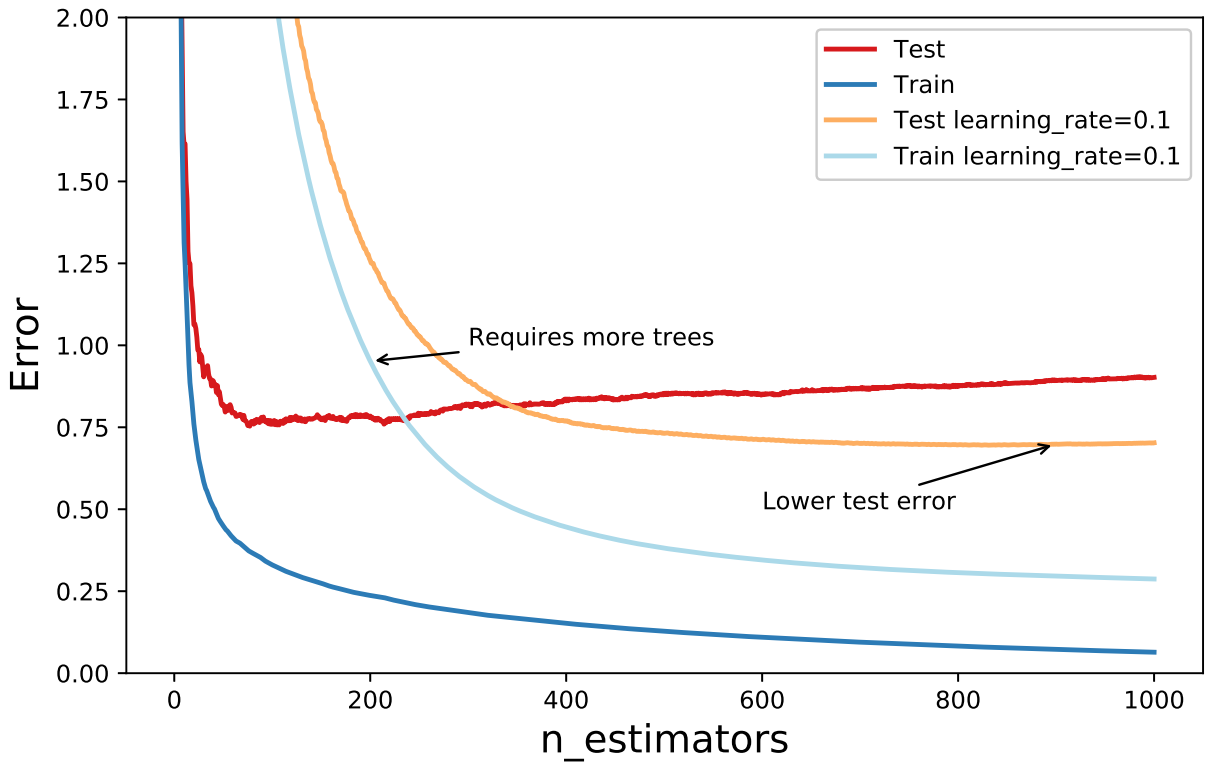
GBM: regularization approaches



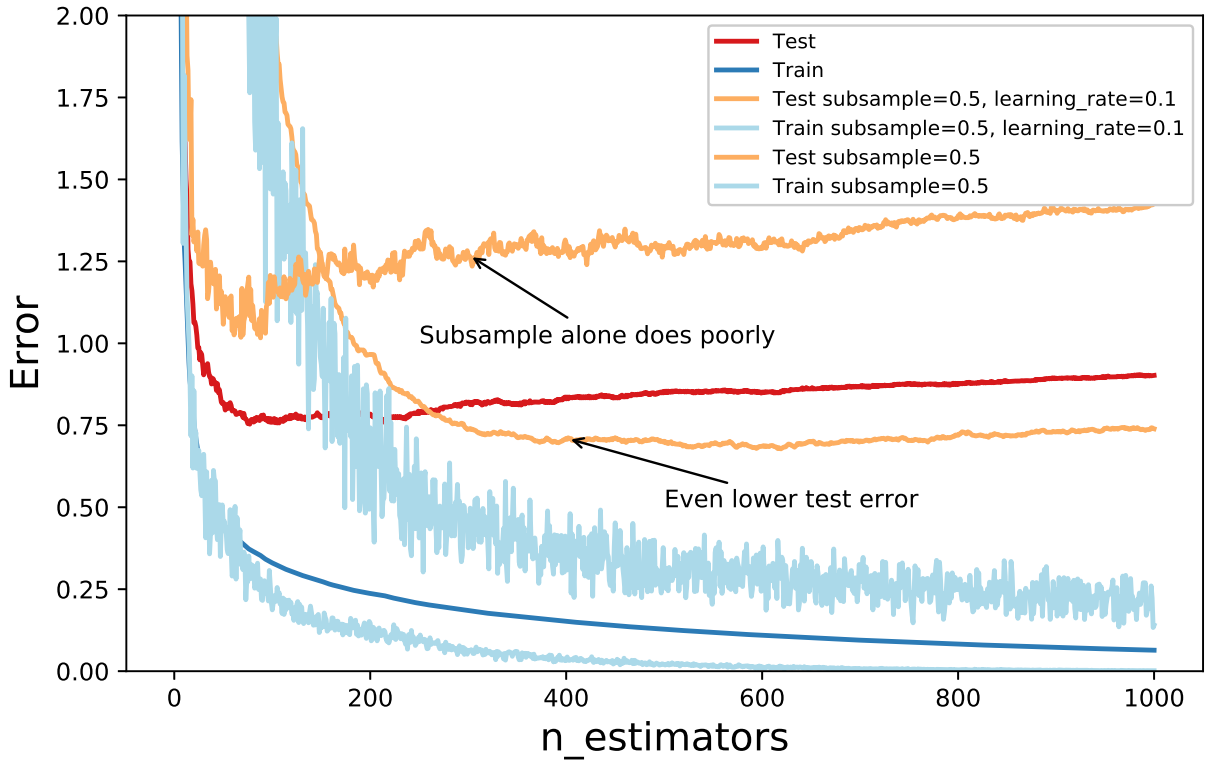
GBM: regularization approaches



GBM: regularization approaches



GBM: regularization approaches



- **Output (response)**: a random variable y
- **Input (explanatory)**: a set of random variables $\mathbf{x} = (x_1, \ldots, x_d)$
- Training sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of $(\mathbf{x}_i, y_i) \sim D$
- Let $f^*(\mathbf{x})$ minimizes the expected loss value $L(y, f(\mathbf{x}))$

$$f^* = \arg\inf_f \mathbb{E}_D L(y, f(\mathbf{x}))$$

- **Goal**: obtain an estimate $\widehat{f}(\mathbf{x})$ of the function $f^*(\mathbf{x})$

- **Output (response)**: a random variable y
- **Input (explanatory)**: a set of random variables $\mathbf{x} = (x_1, \ldots, x_d)$
- Training sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of $(\mathbf{x}_i, y_i) \sim D$
- Let $f^*(\mathbf{x})$ minimizes the expected loss value $L(y, f(\mathbf{x}))$

$$f^* = \arg\inf_f \mathbb{E}_D L(y, f(\mathbf{x}))$$

- **Goal**: obtain an estimate $\widehat{f}(\mathbf{x})$ of the function $f^*(\mathbf{x})$

Function Estimation

- **Output (response)**: a random variable y
- **Input (explanatory)**: a set of random variables $\mathbf{x} = (x_1, \ldots, x_d)$
- Training sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of $(\mathbf{x}_i, y_i) \sim D$
- Let $f^*(\mathbf{x})$ minimizes the expected loss value $L(y, f(\mathbf{x}))$

$$f^* = \arg\inf_f \mathbb{E}_D L(y, f(\mathbf{x}))$$

- **Goal**: obtain an estimate $\widehat{f}(\mathbf{x})$ of the function $f^*(\mathbf{x})$

- **Output (response)**: a random variable y
- **Input (explanatory)**: a set of random variables $\mathbf{x} = (x_1, \ldots, x_d)$
- Training sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of $(\mathbf{x}_i, y_i) \sim D$
- Let $f^*(\mathbf{x})$ minimizes the expected loss value $L(y, f(\mathbf{x}))$

$$f^* = \arg\inf_f \mathbb{E}_D L(y, f(\mathbf{x}))$$

- **Goal**: obtain an estimate $\widehat{f}(\mathbf{x})$ of the function $f^*(\mathbf{x})$

- **Output (response)**: a random variable y
- **Input (explanatory)**: a set of random variables $\mathbf{x} = (x_1, \ldots, x_d)$
- Training sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of $(\mathbf{x}_i, y_i) \sim D$
- Let $f^*(\mathbf{x})$ minimizes the expected loss value $L(y, f(\mathbf{x}))$

$$f^* = \arg\inf_f \mathbb{E}_D L(y, f(\mathbf{x}))$$

- **Goal**: obtain an estimate $\widehat{f}(\mathbf{x})$ of the function $f^*(\mathbf{x})$

Numerical Optimization in Function Space

- We apply numerical optimization in function space
- We
  - consider $f(\mathbf{x})$ evaluated at each point $\mathbf{x}$ as a parameter and
  - seek to minimize

$$\Phi(f) = \mathbb{E}_{y,\mathbf{x}} L(y, f(\mathbf{x})) = \mathbb{E}_{\mathbf{x}} \left[\mathbb{E}_y \, L(y, f(\mathbf{x})) \, | \mathbf{x} \right]$$

    at each individual $\mathbf{x}$, directly with respect to $f(\mathbf{x})$
- Numerical optimization: we approximate $f^*(\mathbf{x})$ by

$$f_T(\mathbf{x}) = \sum_{t=0}^{T} h_t(\mathbf{x}),$$

where $h_0(\mathbf{x})$ is an initial guess, and $\{h_t(\mathbf{x})\}_{t=1}^{T}$ are incremental functions (steps or boosts)

- We apply numerical optimization in function space
- We
  - consider $f(\mathbf{x})$ evaluated at each point $\mathbf{x}$ as a parameter and
  - seek to minimize

$$\Phi(f) = \mathbb{E}_{y,\mathbf{x}} L(y, f(\mathbf{x})) = \mathbb{E}_{\mathbf{x}} \left[\mathbb{E}_y \, L(y, f(\mathbf{x})) \, | \mathbf{x} \right]$$

  at each individual $\mathbf{x}$, directly with respect to $f(\mathbf{x})$
- Numerical optimization: we approximate $f^*(\mathbf{x})$ by

$$f_T(\mathbf{x}) = \sum_{t=0}^{T} h_t(\mathbf{x}),$$

where $h_0(\mathbf{x})$ is an initial guess, and $\{h_t(\mathbf{x})\}_{t=1}^{T}$ are incremental functions (steps or boosts)

- We apply numerical optimization in function space
- We
  - consider $f(\mathbf{x})$ evaluated at each point $\mathbf{x}$ as a parameter and
  - seek to minimize

$$\Phi(f) = \mathbb{E}_{y,\mathbf{x}} L(y, f(\mathbf{x})) = \mathbb{E}_{\mathbf{x}} \left[\mathbb{E}_y \, L(y, f(\mathbf{x})) \, | \mathbf{x} \right]$$

    at each individual $\mathbf{x}$, directly with respect to $f(\mathbf{x})$
- Numerical optimization: we approximate $f^*(\mathbf{x})$ by

$$f_T(\mathbf{x}) = \sum_{t=0}^{T} h_t(\mathbf{x}),$$

where $h_0(\mathbf{x})$ is an initial guess, and $\{h_t(\mathbf{x})\}_{t=1}^{T}$ are incremental functions (steps or boosts)

- Steepest-descent:

$$h_t(\mathbf{x}) = -\alpha_t g_t(\mathbf{x})$$

with

$$g_t(\mathbf{x}) = \left[\frac{\partial \Phi(f(\mathbf{x}))}{\partial f(\mathbf{x})}\right]_{f(\mathbf{x})=f_{t-1}(\mathbf{x})}$$

and

$$f_{t-1}(\mathbf{x}) = \sum_{s=0}^{t-1} h_s(\mathbf{x})$$

- The multiplier α_t is given by the line search

$$\alpha_t = \arg\min_{\alpha} \mathbb{E}_D L(y, f_{t-1}(\mathbf{x}) - \alpha g_t(\mathbf{x}))$$

- Steepest-descent:

$$h_t(\mathbf{x}) = -\alpha_t g_t(\mathbf{x})$$

with

$$g_t(\mathbf{x}) = \left[\frac{\partial \Phi(f(\mathbf{x}))}{\partial f(\mathbf{x})}\right]_{f(\mathbf{x})=f_{t-1}(\mathbf{x})}$$

and

$$f_{t-1}(\mathbf{x}) = \sum_{s=0}^{t-1} h_s(\mathbf{x})$$

- The multiplier α_t is given by the line search

$$\alpha_t = \arg\min_{\alpha} \mathbb{E}_D L(y, f_{t-1}(\mathbf{x}) - \alpha g_t(\mathbf{x}))$$

- Steepest-descent:

$$h_t(\mathbf{x}) = -\alpha_t g_t(\mathbf{x})$$

with

$$g_t(\mathbf{x}) = \left[\frac{\partial \Phi(f(\mathbf{x}))}{\partial f(\mathbf{x})}\right]_{f(\mathbf{x})=f_{t-1}(\mathbf{x})}$$

and

$$f_{t-1}(\mathbf{x}) = \sum_{s=0}^{t-1} h_s(\mathbf{x})$$

- The multiplier α_t is given by the line search

$$\alpha_t = \arg\min_{\alpha} \mathbb{E}_D L(y, f_{t-1}(\mathbf{x}) - \alpha g_t(\mathbf{x}))$$

- Steepest-descent:

$$h_t(\mathbf{x}) = -\alpha_t g_t(\mathbf{x})$$

with

$$g_t(\mathbf{x}) = \left[\frac{\partial \Phi(f(\mathbf{x}))}{\partial f(\mathbf{x})}\right]_{f(\mathbf{x})=f_{t-1}(\mathbf{x})}$$

and

$$f_{t-1}(\mathbf{x}) = \sum_{s=0}^{t-1} h_s(\mathbf{x})$$

- The multiplier α_t is given by the line search

$$\alpha_t = \arg\min_{\alpha} \mathbb{E}_D L(y, f_{t-1}(\mathbf{x}) - \alpha g_t(\mathbf{x}))$$

Idea: let us find such model $h_T(\mathbf{x})$, such that it approximates antigradient $\{-g_T(\mathbf{x}_i)\}_{i=1}^m$

$$h_T = \arg\max_h \sum_{i=1}^m \{(-g_T(\mathbf{x}_i)) - h(\mathbf{x}_i)\}^2$$

Loss Function: $L(y, f) = \frac{1}{2}(y - f)^2$

Algorithm:

1. $f_0(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} y_i$
2. For $t = 1$ to T do
   - $s_i = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x})=f_{t-1}(\mathbf{x})} = y_i - f_{t-1}(\mathbf{x}_i),\ i \in [1, m]$
   - $(\alpha_t, \mathbf{w}_t) = \arg\min_{\mathbf{w},\alpha} \sum_{i=1}^{m} \left[s_i - \alpha h(\mathbf{x}_i; \mathbf{w})\right]^2$
   - $f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \alpha_t h(\mathbf{x}; \mathbf{w}_t)$

- Loss Function: $L(y, F) = |y - f|$,

$$s_i = \text{sign}\left(y_i - f_{t-1}\right)$$

- Consider the special case where each base learner is a J-terminal node regression tree. Each regression tree has the additive form

$$h\left(\mathbf{x}; \mathbf{w} = \{a_j, R_j\}_{j=1}^{J}\right) = \sum_{j=1}^{J} a_j 1\left(\mathbf{x} \in R_j\right)$$

- $\{R_j\}_{j=1}^{J}$ are disjoint regions that collectively cover the space of all values of the predictor variable $\mathbf{x}$

Algorithm:

1. $f_0(\mathbf{x}) = \text{median}\{y_i, i \in [1, m]\}$
2. For $t = 1$ to T do
   - $s_i = \text{sign}\left(y_i - f_{t-1}(\mathbf{x}_i)\right),\ i \in [1, m]$
   - $\{R_{j,t}\}_{j=1}^{J} = J - \text{terminal node tree}\left(\{s_i, \mathbf{x}_i\}_{i=1}^{m}\right)$
   - $w_{j,t} = \text{median}\left\{y_i - f_{t-1}(\mathbf{x}_i),\ \mathbf{x}_i \in R_{j,t}\right\},\ j \in [1, J]$
   - $f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \sum_{j=1}^{J} w_{j,t} 1\left(\mathbf{x} \in R_{j,t}\right)$

Least Absolute Deviation (LAD) Regression III

The LAD algorithm is highly robust:

- Trees use only order information on the individual variables x_k
- Pseudo-responses s_i have only two values, $s_i \in \{-1, +1\}$
- Terminal node updates are based on medians

Other Regression Techniques

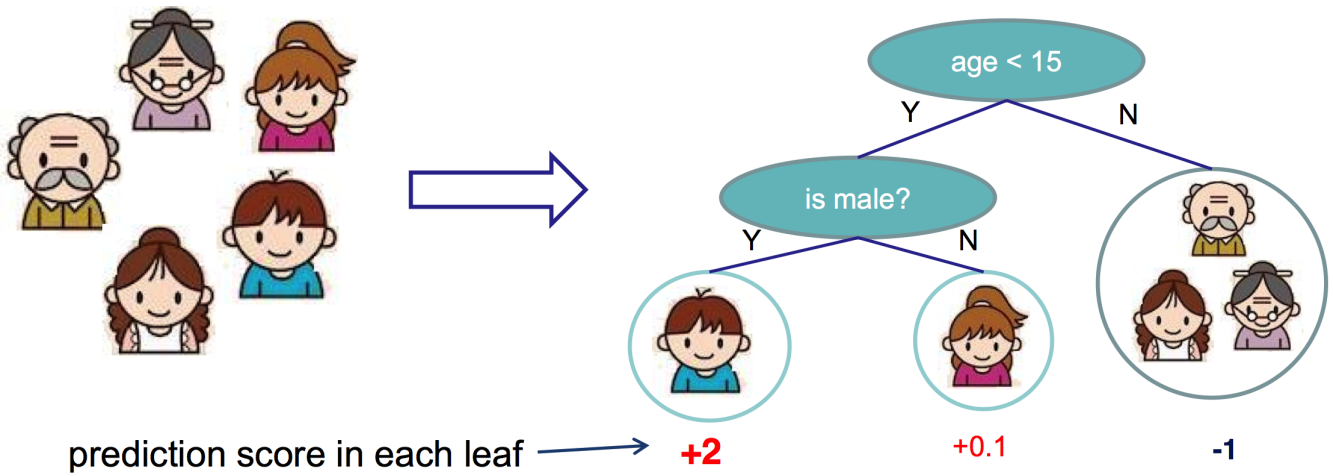
- M-regression
- Two-class logistic regression and classification
- Multiclass logistic regression and classification

Stochastic Gradient Boosting

- On each of boosting steps use some randomly selection subsample
- Usually
  - accuracy is improved
  - convergence is improved
  - learning time is decreased

- Classification and Regression Trees:
  - Decision rules
  - Contains one score in each leaf value

Input: age, gender, occupation,... $\Rightarrow$ Does the person like computer games?



- Model: we have T trees

$$f_T(\mathbf{x}) = \sum_{t=1}^{T} h_t(\mathbf{x}),\ h_t(\mathbf{x}) \in \mathbb{H},$$

  where $\mathbb{H}$ is a space of functions, containing all decision trees
- Parameters: structure of each tree, and the score in the leaf

Learning Process

- Objective: $R(\mathbf{w}) = \sum_{i=1}^{m} L(y_i, \widehat{y}_i) + \sum_{t=1}^{T} \Omega(h_t)$
- We can not use methods such as SGD, to find f_T (since parameters are trees, instead of just numerical vectors) $\rightarrow$ **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$\widehat{y}^{(0)} = h_0(\mathbf{x}) = 0$$
$$\widehat{y}^{(1)} = h_1(\mathbf{x}) = \widehat{y}^{(0)} + h_1(\mathbf{x})$$
$$\widehat{y}^{(2)} = h_1(\mathbf{x}) + h_2(\mathbf{x}) = \widehat{y}^{(1)} + h_2(\mathbf{x})$$
$$\dots$$
$$\widehat{y}^{(t)} = \sum_{s=1}^{t} h_s(\mathbf{x}) = \widehat{y}^{(t-1)} + h_t(\mathbf{x})$$

- Objective: $R(\mathbf{w}) = \sum_{i=1}^{m} L(y_i, \widehat{y}_i) + \sum_{t=1}^{T} \Omega(h_t)$
- We can not use methods such as SGD, to find f_T (since parameters are trees, instead of just numerical vectors) $\rightarrow$ **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$\widehat{y}^{(0)} = h_0(\mathbf{x}) = 0$$
$$\widehat{y}^{(1)} = h_1(\mathbf{x}) = \widehat{y}^{(0)} + h_1(\mathbf{x})$$
$$\widehat{y}^{(2)} = h_1(\mathbf{x}) + h_2(\mathbf{x}) = \widehat{y}^{(1)} + h_2(\mathbf{x})$$
$$\cdots$$
$$\widehat{y}^{(t)} = \sum_{s=1}^{t} h_s(\mathbf{x}) = \widehat{y}^{(t-1)} + h_t(\mathbf{x})$$

- Objective: $R(\mathbf{w}) = \sum_{i=1}^{m} L(y_i, \widehat{y}_i) + \sum_{t=1}^{T} \Omega(h_t)$
- We can not use methods such as SGD, to find f_T (since parameters are trees, instead of just numerical vectors) $\rightarrow$ **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$\widehat{y}^{(0)} = h_0(\mathbf{x}) = 0$$
$$\widehat{y}^{(1)} = h_1(\mathbf{x}) = \widehat{y}^{(0)} + h_1(\mathbf{x})$$
$$\widehat{y}^{(2)} = h_1(\mathbf{x}) + h_2(\mathbf{x}) = \widehat{y}^{(1)} + h_2(\mathbf{x})$$
$$\cdots$$
$$\widehat{y}^{(t)} = \sum_{s=1}^{t} h_s(\mathbf{x}) = \widehat{y}^{(t-1)} + h_t(\mathbf{x})$$

- Objective: $R(\mathbf{w}) = \sum_{i=1}^{m} L(y_i, \widehat{y}_i) + \sum_{t=1}^{T} \Omega(h_t)$
- We can not use methods such as SGD, to find f_T (since parameters are trees, instead of just numerical vectors) $\rightarrow$ **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$\widehat{y}^{(0)} = h_0(\mathbf{x}) = 0$$
$$\widehat{y}^{(1)} = h_1(\mathbf{x}) = \widehat{y}^{(0)} + h_1(\mathbf{x})$$
$$\widehat{y}^{(2)} = h_1(\mathbf{x}) + h_2(\mathbf{x}) = \widehat{y}^{(1)} + h_2(\mathbf{x})$$
$$\dots$$
$$\widehat{y}^{(t)} = \sum_{s=1}^{t} h_s(\mathbf{x}) = \widehat{y}^{(t-1)} + h_t(\mathbf{x})$$

Additive Training

- Iterative Objective Optimization
- At round t we need to tune $h_t(\mathbf{x})$

$$R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t)}\right) + \sum_{s=1}^{t} \Omega(h_s)$$
$$= \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$$

- Consider squared loss

$$R^{(t)} = \sum_{i=1}^{m} \left(y_i - \left(\widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right)\right)^2 + \Omega(h_t) + \text{const}$$
$$= \sum_{i=1}^{m} \left[2\left(\widehat{y}_i^{(t-1)} - y_i\right) h_t(\mathbf{x}_i) + h_t^2(\mathbf{x}_i)\right] + \Omega(h_t) + \text{const}$$

Additive Training

- Iterative Objective Optimization
- At round t we need to tune $h_t(\mathbf{x})$

$$R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t)}\right) + \sum_{s=1}^{t} \Omega(h_s)$$
$$= \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$$

- Consider squared loss

$$R^{(t)} = \sum_{i=1}^{m} \left(y_i - \left(\widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right)\right)^2 + \Omega(h_t) + \text{const}$$
$$= \sum_{i=1}^{m} \left[2\left(\widehat{y}_i^{(t-1)} - y_i\right) h_t(\mathbf{x}_i) + h_t^2(\mathbf{x}_i)\right] + \Omega(h_t) + \text{const}$$

Additive Training

- Iterative Objective Optimization
- At round t we need to tune $h_t(\mathbf{x})$

$$
\begin{aligned}
R^{(t)} &= \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t)}\right) + \sum_{s=1}^{t} \Omega(h_s) \\
&= \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}
\end{aligned}
$$

- Consider squared loss

$$
\begin{aligned}
R^{(t)} &= \sum_{i=1}^{m} \left(y_i - \left(\widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right)\right)^2 + \Omega(h_t) + \text{const} \\
&= \sum_{i=1}^{m} \left[2\left(\widehat{y}_i^{(t-1)} - y_i\right) h_t(\mathbf{x}_i) + h_t^2(\mathbf{x}_i)\right] + \Omega(h_t) + \text{const}
\end{aligned}
$$

Taylor Expansion Approximation of Loss

- Goal: optimize $R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$
  $\rightarrow$ computationally complicated
- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^{m} \left[L\left(y_i, \widehat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) + \text{const}, \quad \text{where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right), \quad d_i = \partial^2_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2\left(\widehat{y}^{(t-1)} - y_i\right),$$

$$d_i = \partial^2_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2$$

Taylor Expansion Approximation of Loss

- Goal: optimize $R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$
  $\rightarrow$ computationally complicated
- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^{m} \left[L\left(y_i, \widehat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) + \text{const}, \quad \text{where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right), \quad d_i = \partial^2_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2\left(\widehat{y}^{(t-1)} - y_i\right),$$

$$d_i = \partial^2_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2$$

Taylor Expansion Approximation of Loss

- Goal: optimize $R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$
  $\rightarrow$ computationally complicated
- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^{m} \left[L\left(y_i, \widehat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) + \text{const}, \quad \text{where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right), \quad d_i = \partial^2_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2\left(\widehat{y}^{(t-1)} - y_i\right),$$

$$d_i = \partial^2_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2$$

Taylor Expansion Approximation of Loss

- Goal: optimize $R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$
  $\rightarrow$ computationally complicated
- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^{m} \left[L\left(y_i, \widehat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) + \text{const}, \quad \text{where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right), \quad d_i = \partial^2_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2\left(\widehat{y}^{(t-1)} - y_i\right),$$

$$d_i = \partial^2_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2$$

Taylor Expansion Approximation of Loss

- Goal: optimize $R^{(t)} = \sum_{i=1}^{m} L\left(y_i, \widehat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)\right) + \Omega(h_t) + \text{const}$
  $\rightarrow$ computationally complicated
- Taylor expansion:

$$R^{(t)} \approx \sum_{i=1}^{m} \left[L\left(y_i, \widehat{y}_i^{(t-1)}\right) + g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) + \text{const}, \quad \text{where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right), \quad d_i = \partial^2_{\widehat{y}^{(t-1)}} L\left(y_i, \widehat{y}^{(t-1)}\right)$$

- In case of a squared loss

$$g_i = \partial_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2\left(\widehat{y}^{(t-1)} - y_i\right),$$

$$d_i = \partial^2_{\widehat{y}^{(t-1)}} \left(\widehat{y}^{(t-1)} - y_i\right)^2 = 2$$

- **Objective**:

$$\sum_{i=1}^{m}\left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i)\right] + \Omega(h_t), \text{ where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right), \; d_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right)$$

- Approximation of the objective depends on $l(\cdot,\cdot)$ only through g_i and d_i. Thus using it we can
  - Computationally efficiently control growing of trees
  - Easily introduce different loss functions $l(\cdot,\cdot)$ inside boosted trees implementation

- **Objective**:

$$\sum_{i=1}^{m}\left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i)\right] + \Omega(h_t), \text{ where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right), \; d_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right)$$

- Approximation of the objective depends on $l(\cdot,\cdot)$ only through g_i and d_i. Thus using it we can
  - Computationally efficiently control growing of trees
  - Easily introduce different loss functions $l(\cdot,\cdot)$ inside boosted trees implementation

- **Objective**:

$$\sum_{i=1}^{m}\left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i)\right] + \Omega(h_t), \text{ where}$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right), \; d_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right)$$

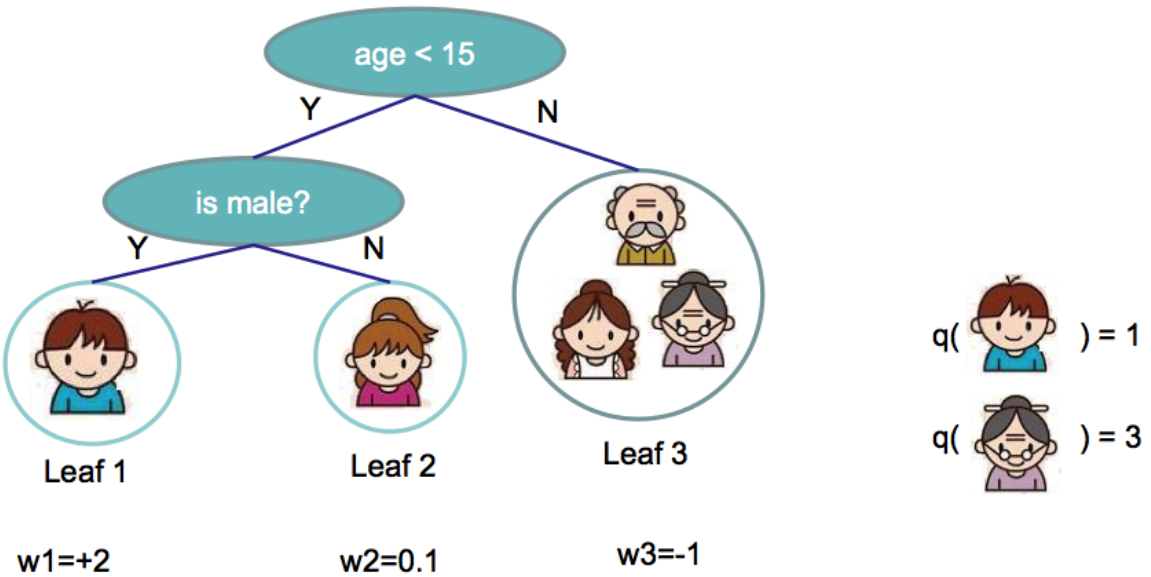
- Approximation of the objective depends on $l(\cdot, \cdot)$ only through g_i and d_i. Thus using it we can
  - Computationally efficiently control growing of trees
  - Easily introduce different loss functions $l(\cdot, \cdot)$ inside boosted trees implementation

- We define a tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$h_t(\mathbf{x}) = w_{q(\mathbf{x})},$$

where

- $\mathbf{w} = (w_1, \ldots, w_J) \in \mathbb{R}^J$ are the leaf weights of the tree
- $q : \mathbf{x} \to \{1, 2, \ldots, J\}$ is the structure of the tree

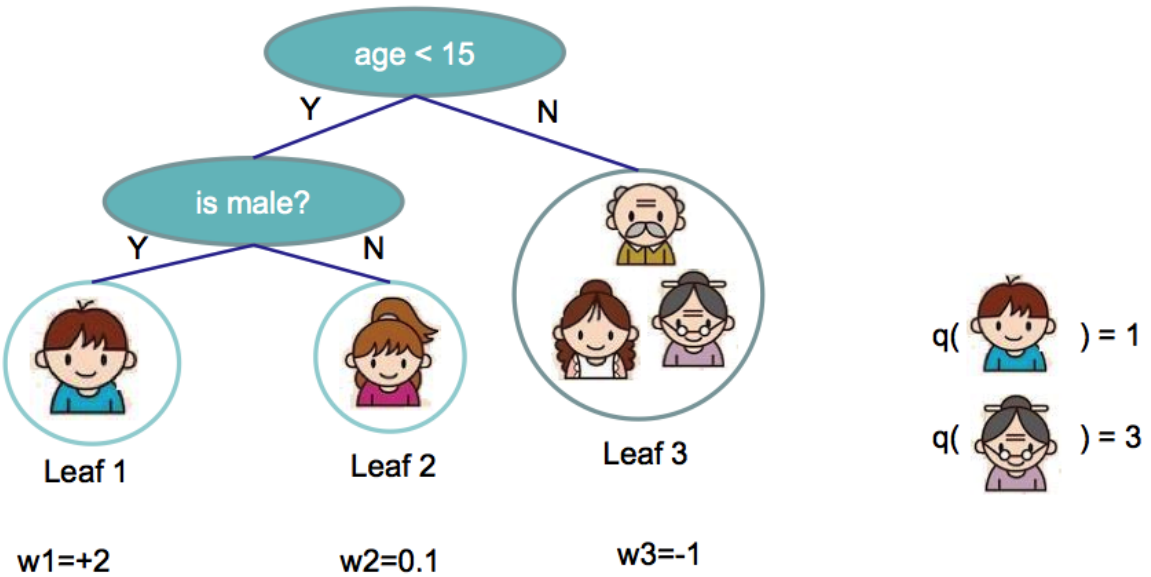


- We define a tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$h_t(\mathbf{x}) = w_{q(\mathbf{x})},$$

where

- $\mathbf{w} = (w_1, \ldots, w_J) \in \mathbb{R}^J$ are the leaf weights of the tree
- $q : \mathbf{x} \to \{1, 2, \ldots, J\}$ is the structure of the tree



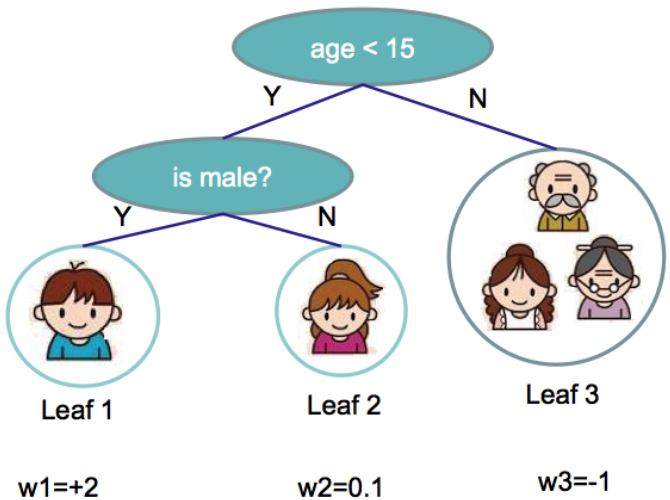
Complexity of a Tree

- Define complexity as (this is not the only possible definition)

$$\Omega(h_t) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \omega_j^2,$$

where

- J is a number of leaves in h_t
- L_2-norm of leaf scores in h_t is used as a regularizer



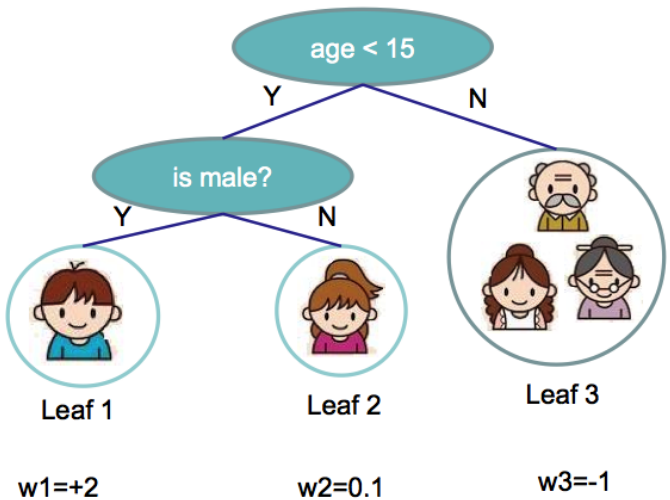
$$\Omega = \gamma \cdot 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

- Define complexity as (this is not the only possible definition)

$$\Omega(h_t) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \omega_j^2,$$

where

- J is a number of leaves in h_t
- L_2-norm of leaf scores in h_t is used as a regularizer



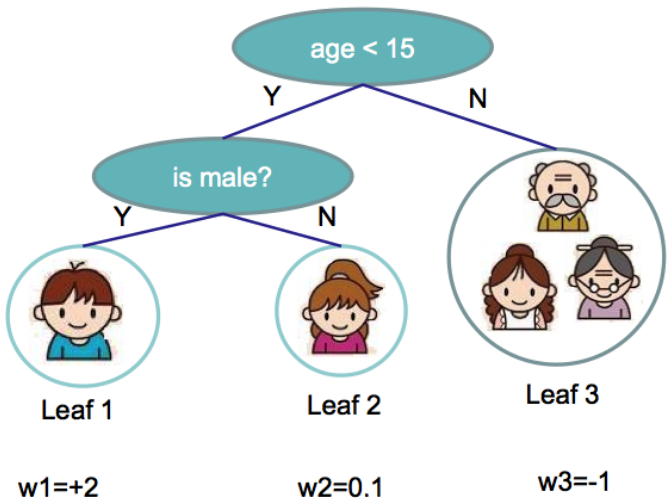
$$\Omega = \gamma \cdot 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

- Define complexity as (this is not the only possible definition)

$$\Omega(h_t) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \omega_j^2,$$

where

- J is a number of leaves in h_t
- L_2-norm of leaf scores in h_t is used as a regularizer



$$\Omega = \gamma \cdot 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Re-define the Objectives

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$
\begin{aligned}
R^{(t)} &\approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) \\
&= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2 \\
&= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J
\end{aligned}
$$

- This is a sum of J independent quadratic functions

Re-define the Objectives

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$
\begin{aligned}
R^{(t)} &\approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) \\
&= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2 \\
&= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J
\end{aligned}
$$

- This is a sum of J independent quadratic functions

Re-define the Objectives

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$R^{(t)} \approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t)$$

$$= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2$$

$$= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J$$

- This is a sum of J independent quadratic functions

Re-define the Objectives

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$R^{(t)} \approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t)$$

$$= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2$$

$$= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J$$

- This is a sum of J independent quadratic functions

Re-define the Objectives

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$
\begin{aligned}
R^{(t)} &\approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) \\
&= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2 \\
&= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J
\end{aligned}
$$

- This is a sum of J independent quadratic functions

- Define the instance set in the leaf j as $I_j = \{i | \, q(\mathbf{x}_i) = j\}$
- Re-group the objective by each leaf

$$
\begin{aligned}
R^{(t)} &\approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t) \\
&= \sum_{i=1}^{m} \left[g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} d_i w_{q(\mathbf{x}_i)}^2 \right] + \gamma J + \lambda \frac{1}{2} \sum_{j=1}^{J} w_j^2 \\
&= \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} d_i + \lambda \right) w_j^2 \right] + \gamma J
\end{aligned}
$$

- This is a sum of J independent quadratic functions

Explicit Solution

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} \left(D_j + \lambda \right) w_j^2 \right] + \gamma J$$

- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \; R^{(t)} = \underbrace{-\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term (*) measure how good a tree structure is

Explicit Solution

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} \left(D_j + \lambda \right) w_j^2 \right] + \gamma J$$

- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \; R^{(t)} = \underbrace{-\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term (*) measure how good a tree structure is

Explicit Solution

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} \left(D_j + \lambda \right) w_j^2 \right] + \gamma J$$

- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \; R^{(t)} = \underbrace{-\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term (*) measure how good a tree structure is

Explicit Solution

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} \left(D_j + \lambda \right) w_j^2 \right] + \gamma J$$

- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \; R^{(t)} = \underbrace{-\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term (*) measure how good a tree structure is

Explicit Solution

- Let us define $G_j = \sum_{i \in I_j} g_i$, $D_j = \sum_{i \in I_j} d_i$

$$R^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} \left(D_j + \lambda \right) w_j^2 \right] + \gamma J$$

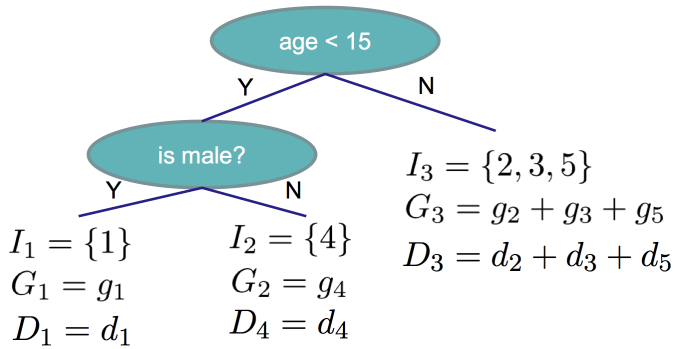
- Assume the structure of the tree (defined by the function $q(\mathbf{x})$) is fixed, then the optimal weight in each leaf, and the resulting objective value are

$$\omega_j^* = -\frac{G_j}{D_j + \lambda}, \; R^{(t)} = \underbrace{-\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{D_j + \lambda}}_{(*)} + \gamma J$$

- The term (*) measure how good a tree structure is

Calculation of the Objective

Instance index		gradient statistics
1		g1, d1
2		g2, d2
3		g3, d3
4		g4, d4
5		g5, d5



$$R^{(t)} = -\frac{1}{2} \sum_j \frac{G_j^2}{D_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

- Enumerate the possible tree structures by $q(\mathbf{x})$
- Calculate the structure score for the $q(\mathbf{x})$, using the scoring equation

$$R^{(t)} = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_j^2}{D_j + \lambda} + \gamma J$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{D_j + \lambda}$$

- However, there can be infinite possible tree structures

Searching Algorithm for Single Tree

- Enumerate the possible tree structures by $q(\mathbf{x})$
- Calculate the structure score for the $q(\mathbf{x})$, using the scoring equation

$$R^{(t)} = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_j^2}{D_j + \lambda} + \gamma J$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{D_j + \lambda}$$

- However, there can be infinite possible tree structures

Searching Algorithm for Single Tree

- Enumerate the possible tree structures by $q(\mathbf{x})$
- Calculate the structure score for the $q(\mathbf{x})$, using the scoring equation

$$R^{(t)} = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_j^2}{D_j+\lambda} + \gamma J$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{D_j+\lambda}$$

- However, there can be infinite possible tree structures

- Enumerate the possible tree structures by $q(\mathbf{x})$
- Calculate the structure score for the $q(\mathbf{x})$, using the scoring equation

$$R^{(t)} = -\frac{1}{2}\sum_{j=1}^{J}\frac{G_j^2}{D_j+\lambda} + \gamma J$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{D_j+\lambda}$$

- However, there can be infinite possible tree structures

In practice, we grow the tree greedily

- Start from a tree with depth 0
- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$\Delta R^{(t)} = \underbrace{\frac{G_L^2}{D_L + \lambda}}_{(1)} + \underbrace{\frac{G_R^2}{D_R + \lambda}}_{(2)} - \underbrace{\frac{(G_L + G_R)^2}{D_L + D_R + \lambda}}_{(3)} - \underbrace{\gamma}_{(4)},$$

  where
  - (1) and (2) are scores of Left and Right childs,
  - (3) is the score if we do not split
  - (4) is the complexity cost by introducing additional leaf
- How do we find the best split?

In practice, we grow the tree greedily

- Start from a tree with depth 0
- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$\Delta R^{(t)} = \underbrace{\frac{G_L^2}{D_L + \lambda}}_{(1)} + \underbrace{\frac{G_R^2}{D_R + \lambda}}_{(2)} - \underbrace{\frac{(G_L + G_R)^2}{D_L + D_R + \lambda}}_{(3)} - \underbrace{\gamma}_{(4)},$$

  where
  - (1) and (2) are scores of Left and Right childs,
  - (3) is the score if we do not split
  - (4) is the complexity cost by introducing additional leaf
- How do we find the best split?

In practice, we grow the tree greedily

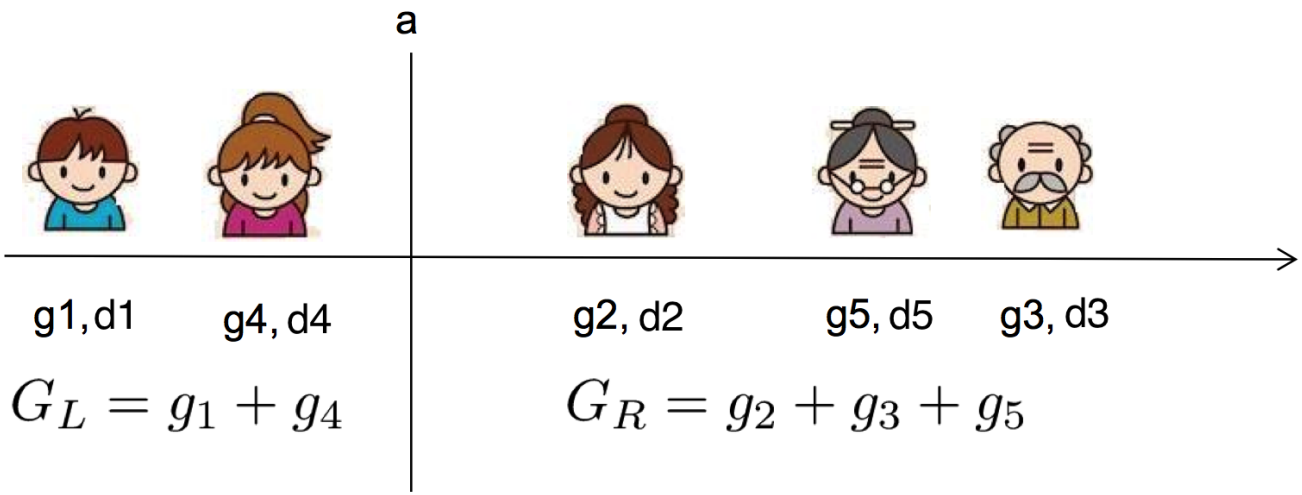
- Start from a tree with depth 0
- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$\Delta R^{(t)} = \underbrace{\frac{G_L^2}{D_L + \lambda}}_{(1)} + \underbrace{\frac{G_R^2}{D_R + \lambda}}_{(2)} - \underbrace{\frac{(G_L + G_R)^2}{D_L + D_R + \lambda}}_{(3)} - \underbrace{\gamma}_{(4)},$$

  where
  - (1) and (2) are scores of Left and Right childs,
  - (3) is the score if we do not split
  - (4) is the complexity cost by introducing additional leaf
- How do we find the best split?

Efficient Finding of the Best Split

- What is the gain of a split rule $x_k < a$? E.g. x_k is an age



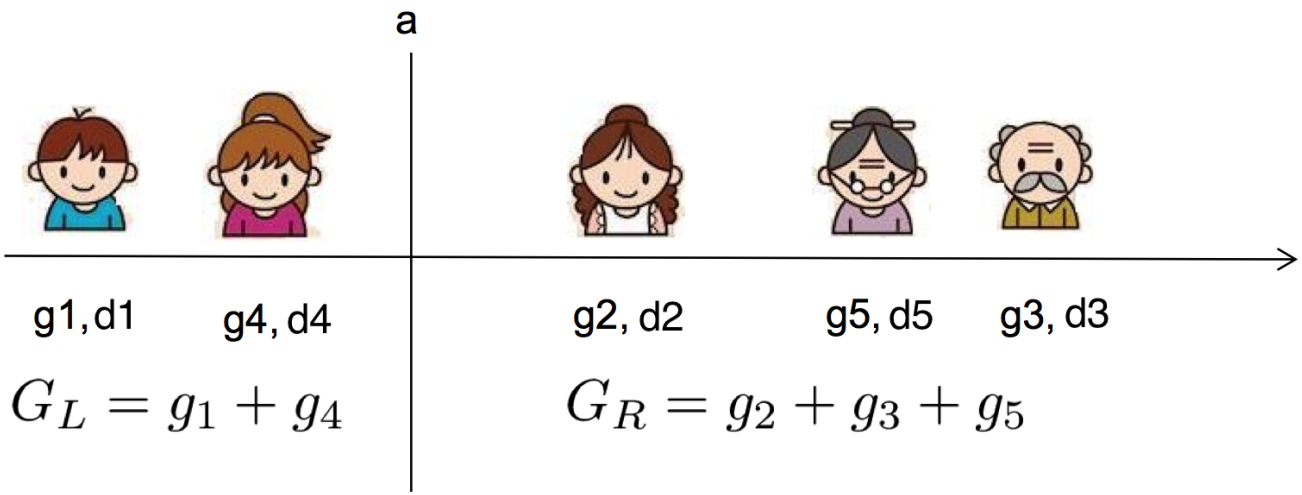
- We sum g's and d's in each leaf and calculate

$$\Delta R^{(t)} = \frac{G_L^2}{D_L + \lambda} + \frac{G_R^2}{D_R + \lambda} - \frac{(G_L + G_R)^2}{D_L + D_R + \lambda} - \gamma,$$

- Left to right linear scan over sorted instances is enough to decide the best split along the feature x_k

Efficient Finding of the Best Split

- What is the gain of a split rule $x_k < a$? E.g. x_k is an age

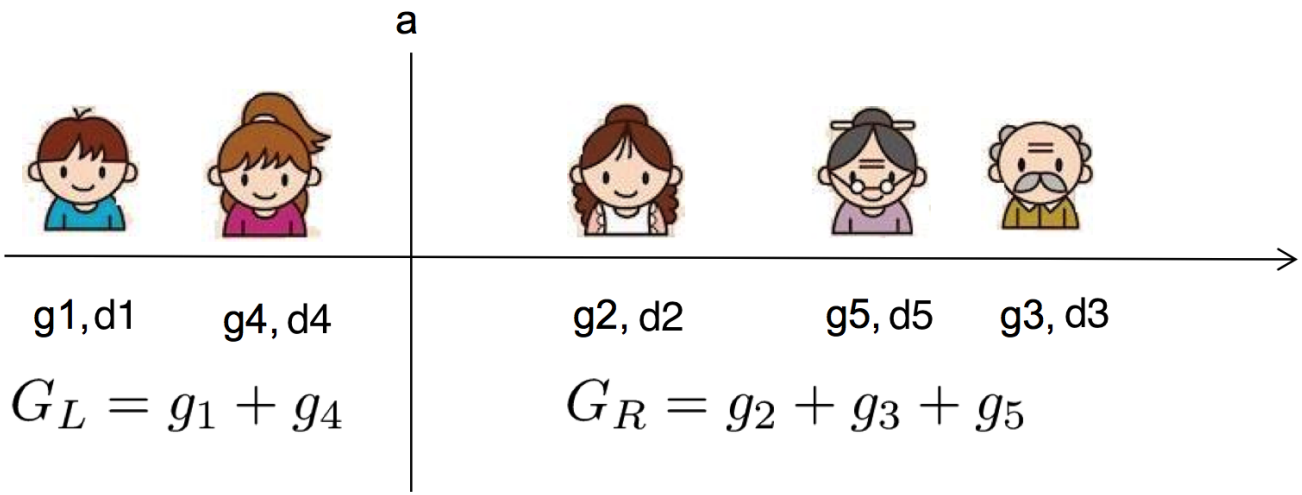


- We sum g's and d's in each leaf and calculate

$$\Delta R^{(t)} = \frac{G_L^2}{D_L + \lambda} + \frac{G_R^2}{D_R + \lambda} - \frac{(G_L + G_R)^2}{D_L + D_R + \lambda} - \gamma,$$

- Left to right linear scan over sorted instances is enough to decide the best split along the feature x_k

- What is the gain of a split rule $x_k < a$? E.g. x_k is an age



- We sum g's and d's in each leaf and calculate

$$\Delta R^{(t)} = \frac{G_L^2}{D_L + \lambda} + \frac{G_R^2}{D_R + \lambda} - \frac{(G_L + G_R)^2}{D_L + D_R + \lambda} - \gamma,$$

- Left to right linear scan over sorted instances is enough to decide the best split along the feature x_k

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion

Conclusions

- The Gradient Boosting Machine: a general approach to boosting adding weak learners that approximate gradient of the loss function
- Gradient Boosting is the most general
  - any loss function
  - good for classification, regression, ranking
- Usually Stochastic gradient boosting is more accurate and faster
- GB is usually applied to decision trees
- In practice GB works only with big T and weak learners (small T and strong learner are not efficient)
- AdaBoost: gradient boosting with an exponential loss function resulting in reweighting training instances when adding weak learners
- XGBoost: gradient boosting with second order optimization, penalized loss and particular choice of impurity criterion