Ensemble and Boosting Algorithms

Weinan Zhang

Shanghai Jiao Tong University

http://wnzhang.net

http://wnzhang.net/teaching/cs420/index.html

Content of this lecture

- Ensemble Methods

- Bagging

- Random Forest

- AdaBoost

- Gradient Boosting Decision Trees

Content of this lecture

- **Ensemble Methods**

- Bagging

- Random Forest

- AdaBoost

- Gradient Boosting Decision Trees

Ensemble Learning

- Consider a set of predictors $f_1, ..., f_L$
  - Different predictors have different performance across data
- Idea: construct a predictor $F(x)$ that combines the individual decisions of $f_1, ..., f_L$
  - E.g., could have the member predictor vote
  - E.g., could use different members for different region of the data space
  - Works well if the member each has low error rate
- Successful ensembles require diversity
  - Predictors should make different mistakes
  - Encourage to involve different types of predictors

Ensemble Learning

Single model
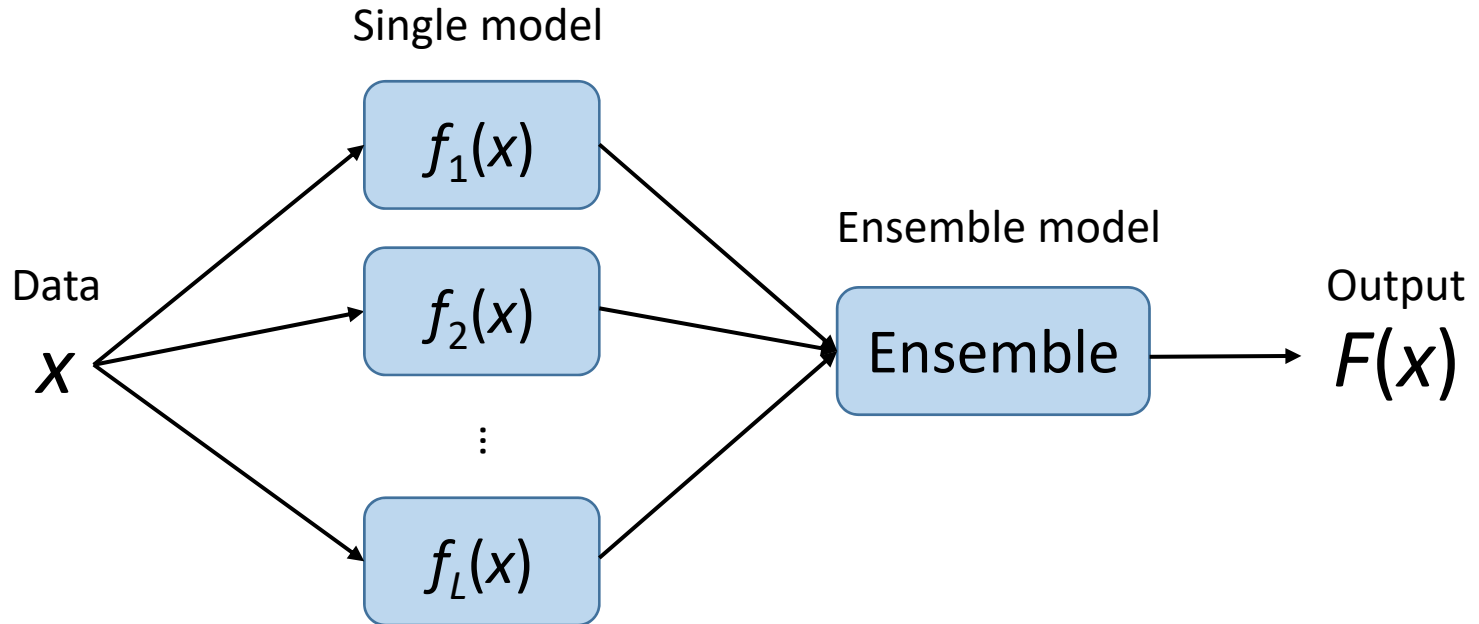
- Although complex, ensemble learning probably offers the most sophisticated output and the best empirical performance!

Practical Application in Competitions

- Netflix Prize Competition
  - Task: predict the user's rating on a movie, given some users' ratings on some movies
  - Called 'collaborative filtering' (we will have a lecture about it later)

- Winner solution
  - BellKor's Pragmatic Chaos – an ensemble of more than 800 predictors

Yehuda Koren

[Yehuda Koren. The BellKor Solution to the Netflix Grand Prize. 2009.]

Practical Application in Competitions

- KDD-Cup 2011 Yahoo! Music Recommendation
  - Task: predict the user's rating on a music, given some users' ratings on some music
    - With music information like album, artist, genre IDs
- Winner solution
  - From A graduate course of National Taiwan University - an ensemble of 221 predictors

A Linear Ensemble of Individual and Blended Models for Music Rating Prediction

Po-Lung Chen, Chen-Tse Tsai, Yao-Nan Chen, Ku-Chun Chou, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Yu-Cheng Chou, Chung-Yi Li, Wei-Shih Lin, Shu-Hao Yu, Rong-Bing Chiu, Chieh-Yen Lin, Chien-Chih Wang, Po-Wei Wang, Wei-Lun Su, Chen-Hung Wu, Tsung-Ting Kuo, Todd G. McKenzie, Ya-Hsuan Chang, Chun-Sung Ferng, Chia-Mau Ni, Hsuan-Tien Lin, Chih-Jen Lin, Shou-De Lin

{R99922038, R98922028, R99922008, R99922095, B97018, B97705004, B96018, B96115, B96069, B96113, B95076, B97114, B97042, D98922007, B97058, B96110, B96055, D97944007, D97041, B96025, R99922054, B96092, HTLIN, CJLIN, SDLIN}@CSIE.NTU.EDU.TW

Department of Computer Science and Information Engineering,
National Taiwan University

Practical Application in Competitions

- KDD-Cup 2011 Yahoo! Music Recommendation
  - Task: predict the user's rating on a music, given some users' ratings on some music
    - With music information like album, artist, genre IDs

- 3rd place solution
  - SJTU-HKUST joint team, an ensemble of 16 predictors

Informative Ensemble of Multi-Resolution Dynamic Factorization Models

Tianqi Chen[*], Zhao Zheng, Qiuxia Lu, Xiao Jiang, Yuqiang Chen, Weinan Zhang
Kailong Chen and Yong Yu
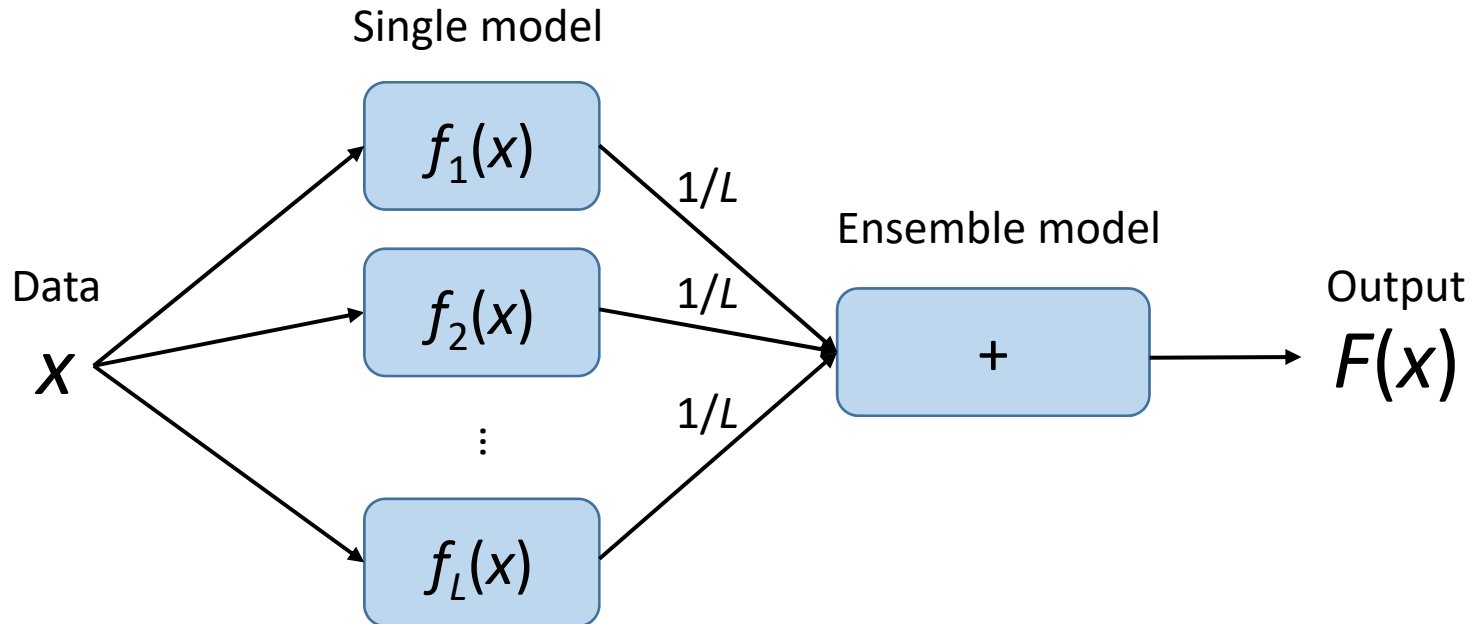Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240 China
{tqchen, zhengzhao,luqiuxia,jiangxiao,yuqiangchen,wnzhang,chenkl,yyu}@apex.sjtu.edu.cn

Nathan N. Liu[†], Bin Cao, Luheng He and Qiang Yang
Hong Kong University of Science and Technology
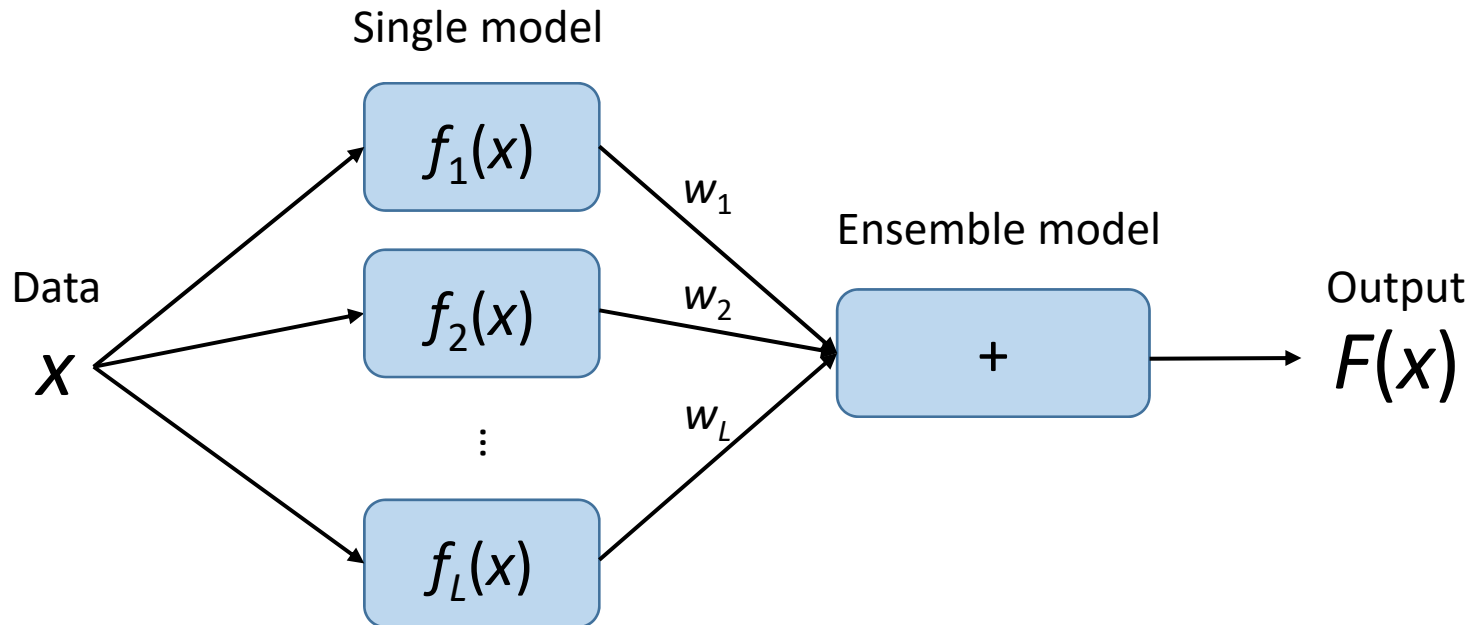Clear Water Bay, Hong Kong
{nliu,caobin,luhenghe,qyang}@cse.ust.hk

Combining Predictor: Averaging

Single model

$$F(x) = \frac{1}{L} \sum_{i=1}^{L} f_i(x)$$
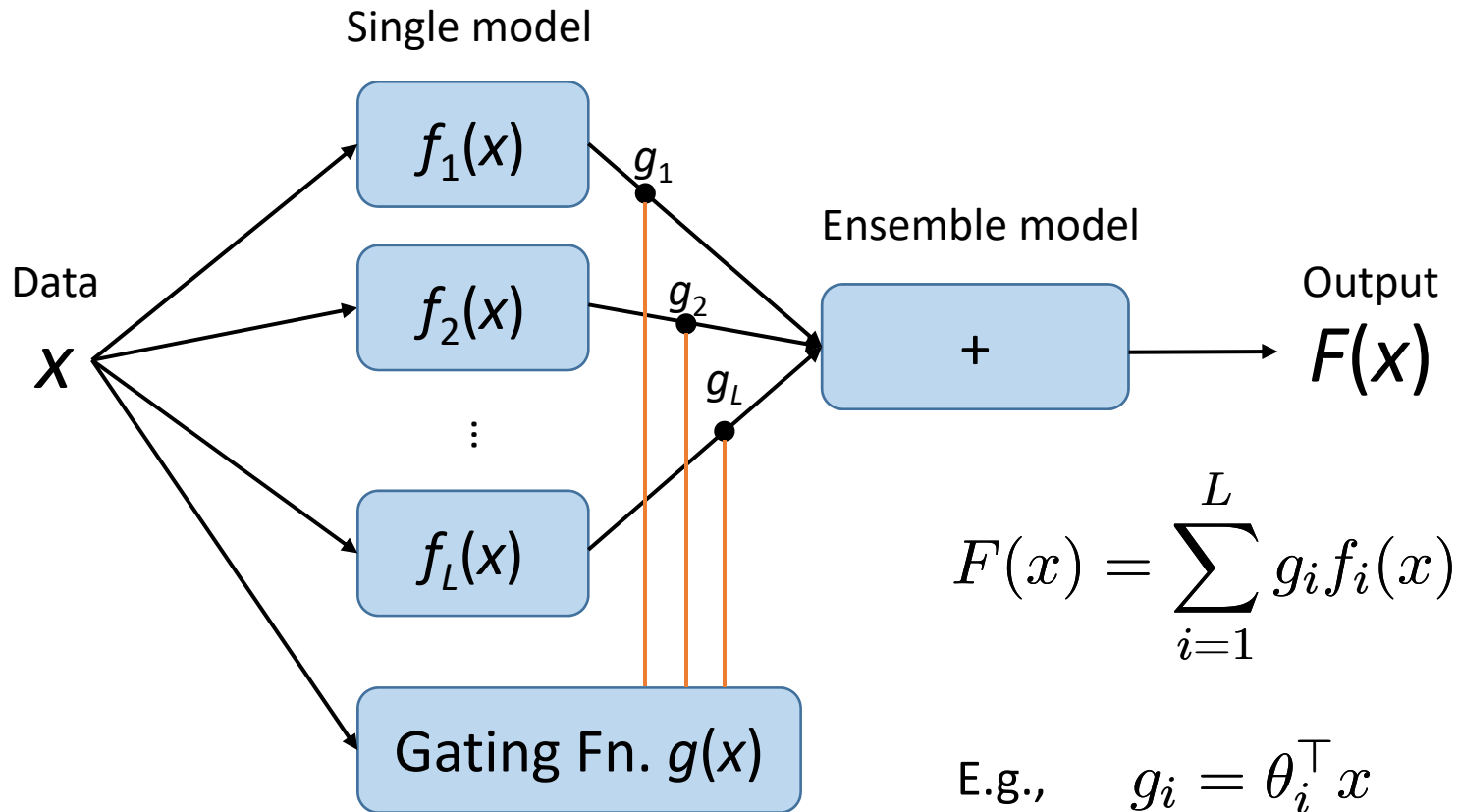
- Averaging for regression; voting for classification

Combining Predictor: Weighted Avg

Single model

Data

x

Ensemble model

Output

$F(x)$

$f_1(x)$

w_1

$f_2(x)$

w_2

w_L

$f_L(x)$

$+$

$$F(x) = \sum_{i=1}^{L} w_i f_i(x)$$

- Just like linear regression or classification
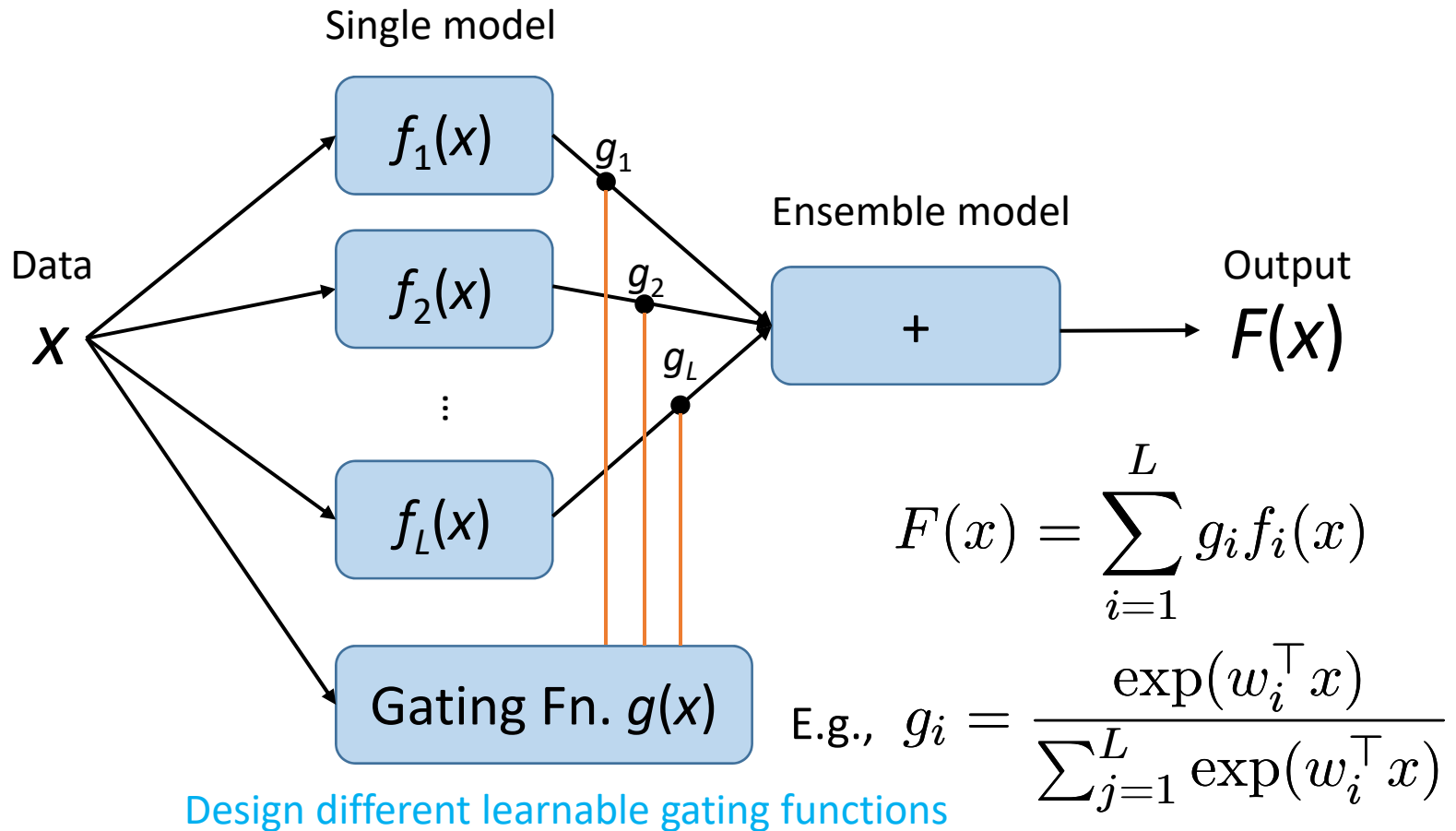- Note: single model will not be updated when training ensemble model

Combining Predictor: Gating

Single model

$$F(x) = \sum_{i=1}^{L} g_i f_i(x)$$

E.g., $\quad g_i = \theta_i^\top x$

Design different learnable gating functions

- Just like linear regression or classification
- Note: single model will not be updated when training ensemble model

Combining Predictor: Gating

Single model

$f_1(x)$ g_1

Data   Ensemble model   Output

x $f_2(x)$ g_2 + $F(x)$

g_L

$f_L(x)$

$$F(x) = \sum_{i=1}^{L} g_i f_i(x)$$

Gating Fn. $g(x)$ E.g., $g_i = \dfrac{\exp(w_i^\top x)}{\sum_{j=1}^{L} \exp(w_i^\top x)}$

Design different learnable gating functions

- Just like linear regression or classification
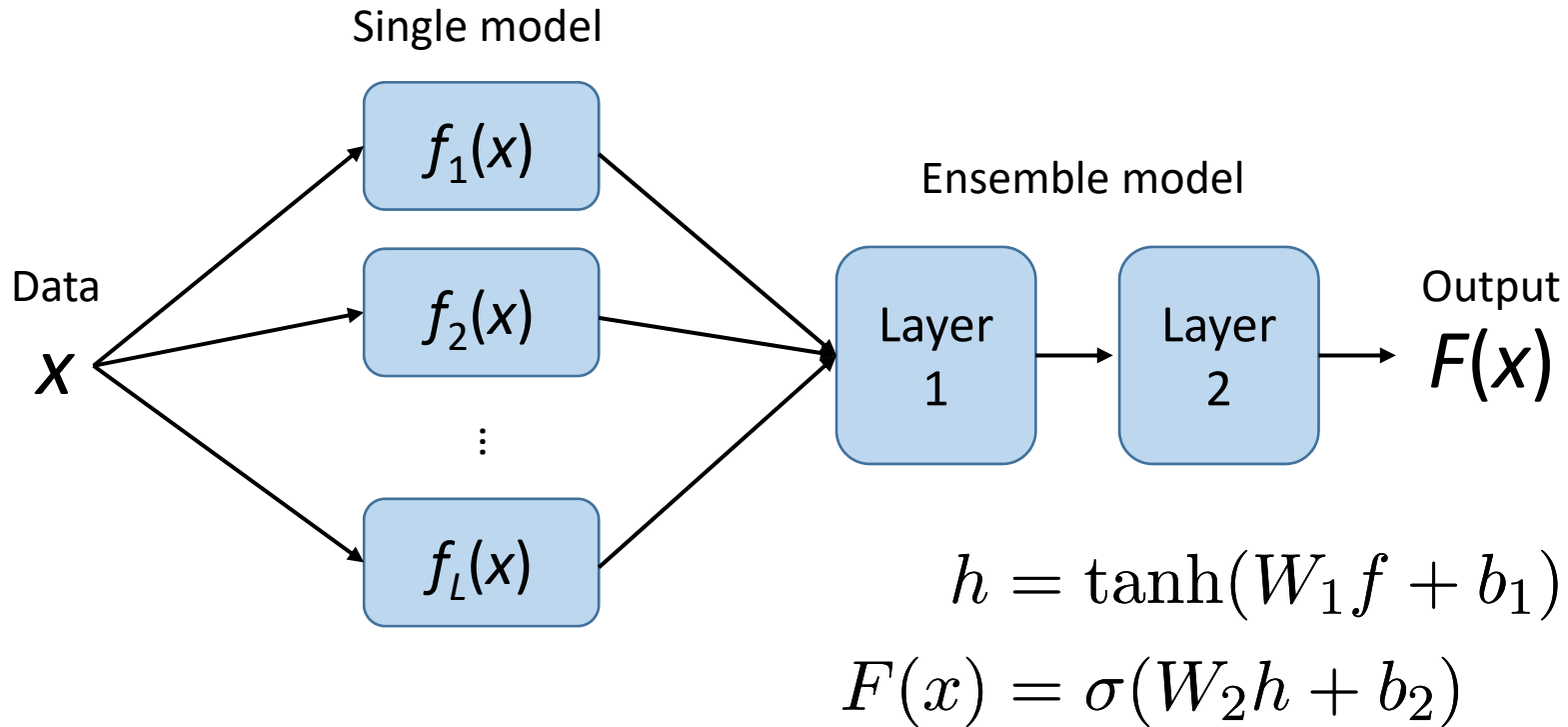- Note: single model will not be updated when training ensemble model

Combining Predictor: Stacking
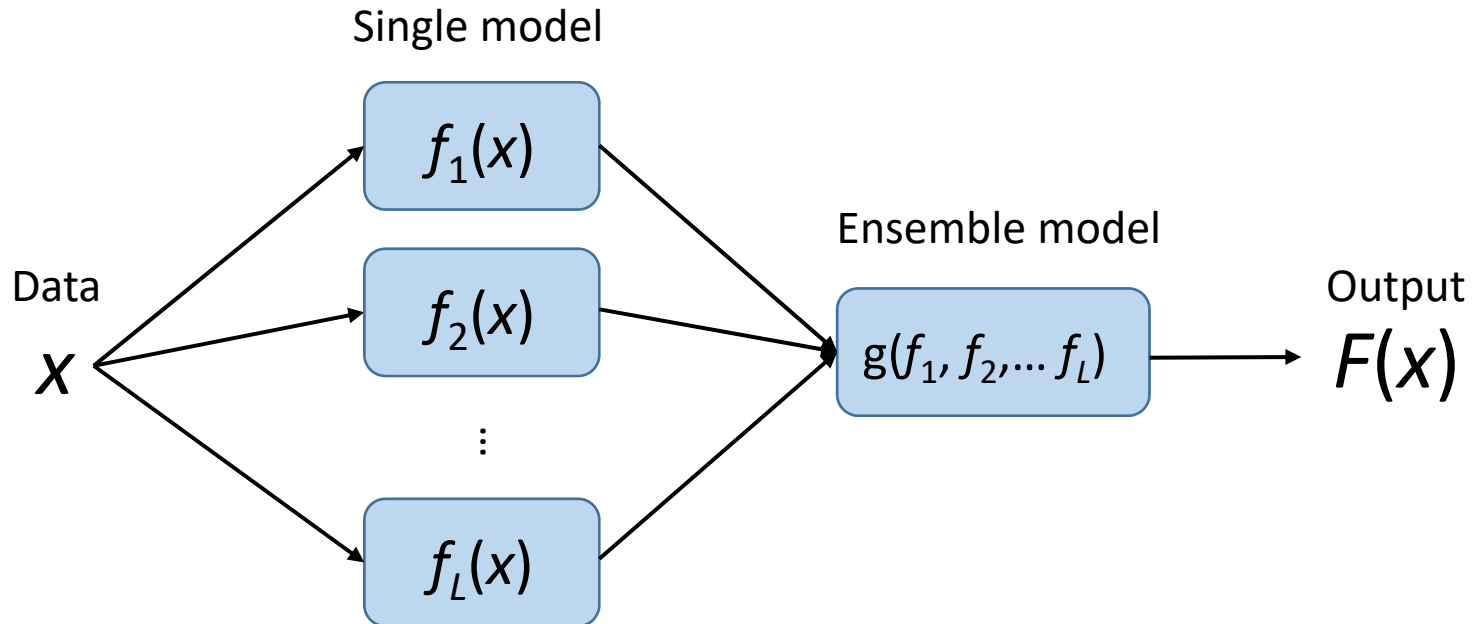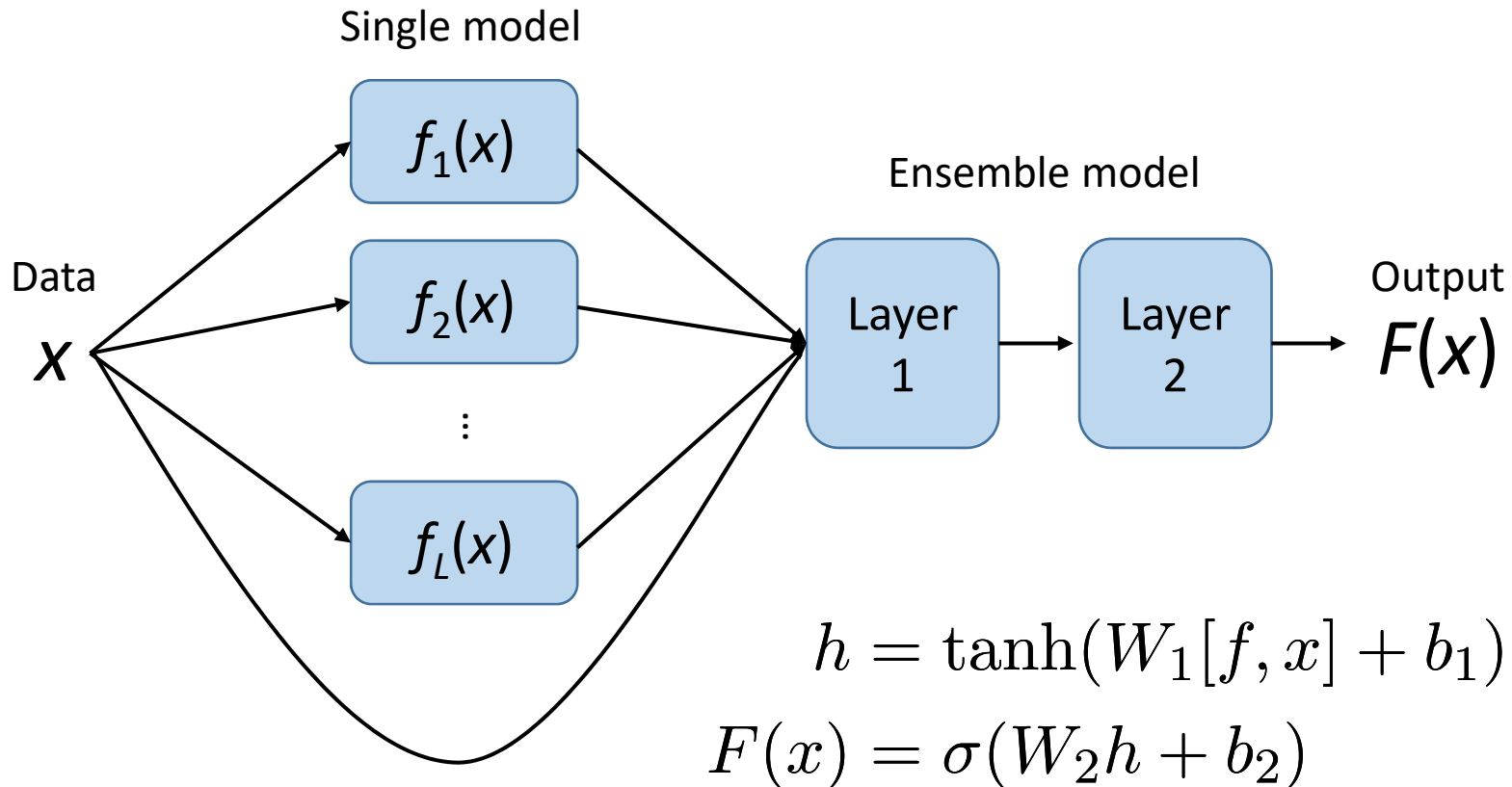

$$F(x) = g(f_1(x), f_2(x), \ldots, f_L(x))$$

- This is the general formulation of an ensemble
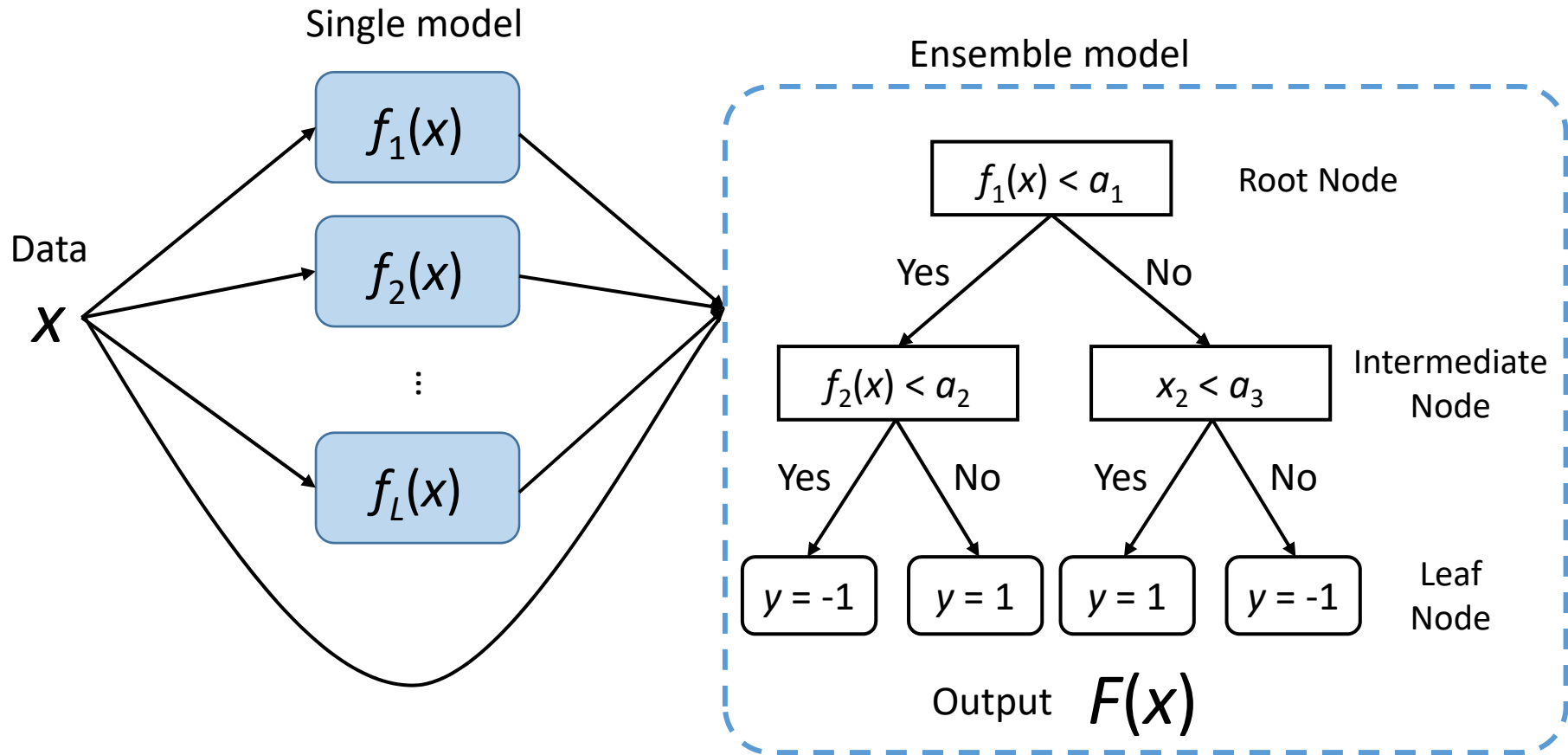
Combining Predictor: Multi-Layer

Single model

$$h = \tanh(W_1 f + b_1)$$

$$F(x) = \sigma(W_2 h + b_2)$$

- Use neural networks as the ensemble model

Combining Predictor: Multi-Layer

Single model

$f_1(x)$

Ensemble model

Data

x

$f_2(x)$

Output

Layer 1 → Layer 2 → $F(x)$

⋮

$f_L(x)$

$$h = \tanh(W_1[f, x] + b_1)$$

$$F(x) = \sigma(W_2 h + b_2)$$

- Use neural networks as the ensemble model
- Incorporate *x* into the first hidden layer (as gating)

Combining Predictor: Tree Models

Single model

Ensemble model

Data

x

$f_1(x)$

$f_2(x)$

$\vdots$

$f_L(x)$

$f_1(x) < a_1$ — Root Node

Yes   No

$f_2(x) < a_2$ $x_2 < a_3$ — Intermediate Node

Yes   No   Yes   No

$y = -1$ $y = 1$ $y = 1$ $y = -1$ — Leaf Node

Output $F(x)$

- Use decision trees as the ensemble model
- Splitting according to the value of f's and x

Diversity for Ensemble Input

- Successful ensembles require diversity
  - Predictors may make different mistakes
  - Encourage to
    - involve different types of predictors
    - vary the training sets
    - vary the feature sets

Cause of the Mistake	Diversification Strategy
Pattern was difficult	Try different models
Overfitting	Vary the training sets
Some features are noisy	Vary the set of input features

[Based on slide by Leon Bottou]

Content of this lecture

- Ensemble Methods

- Bagging

- Random Forest

- AdaBoost

- Gradient Boosting Decision Trees

Manipulating the Training Data

- Bootstrap replication
  - Given *n* training samples *Z*, construct a new training set *Z** by sampling *n* instances with replacement
  - Excludes about 37% of the training instances

$$P\{\text{observation } i \in \text{bootstrap samples}\} = 1 - \left(1 - \frac{1}{N}\right)^N$$
$$\simeq 1 - e^{-1} = 0.632$$

- Bagging (Bootstrap Aggregating)
  - Create bootstrap replicates of training set
  - Train a predictor for each replicate
  - Validate the predictor using out-of-bootstrap data
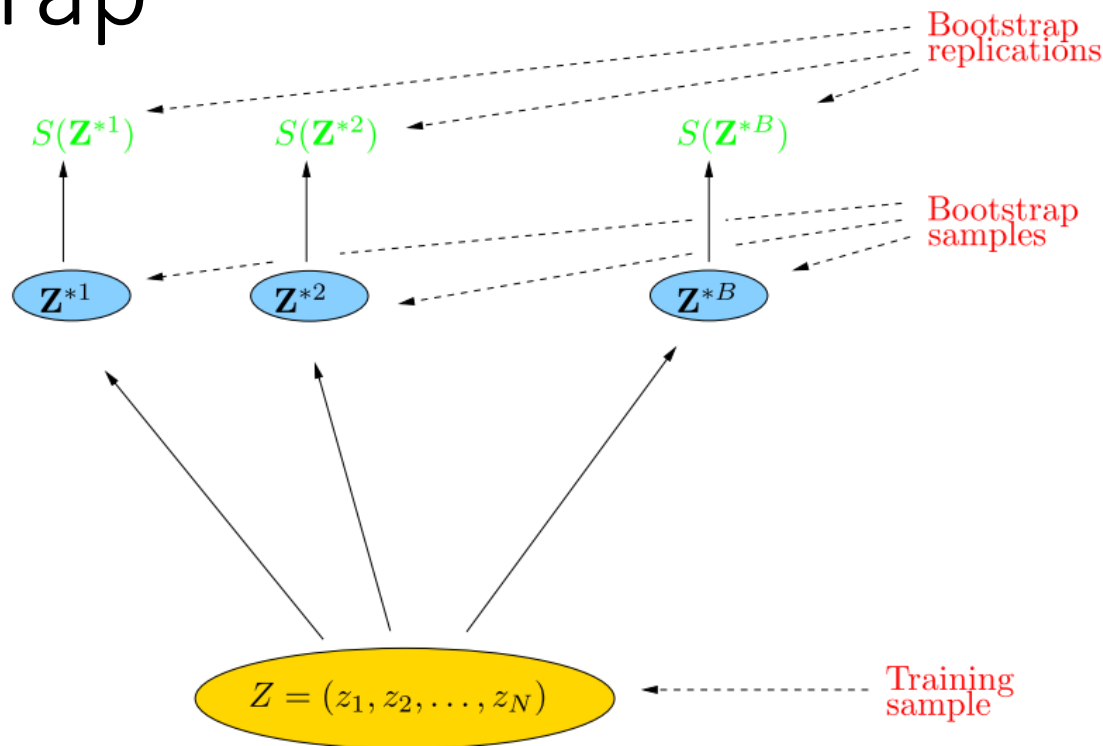 - Average output of all predictors

Bootstrap
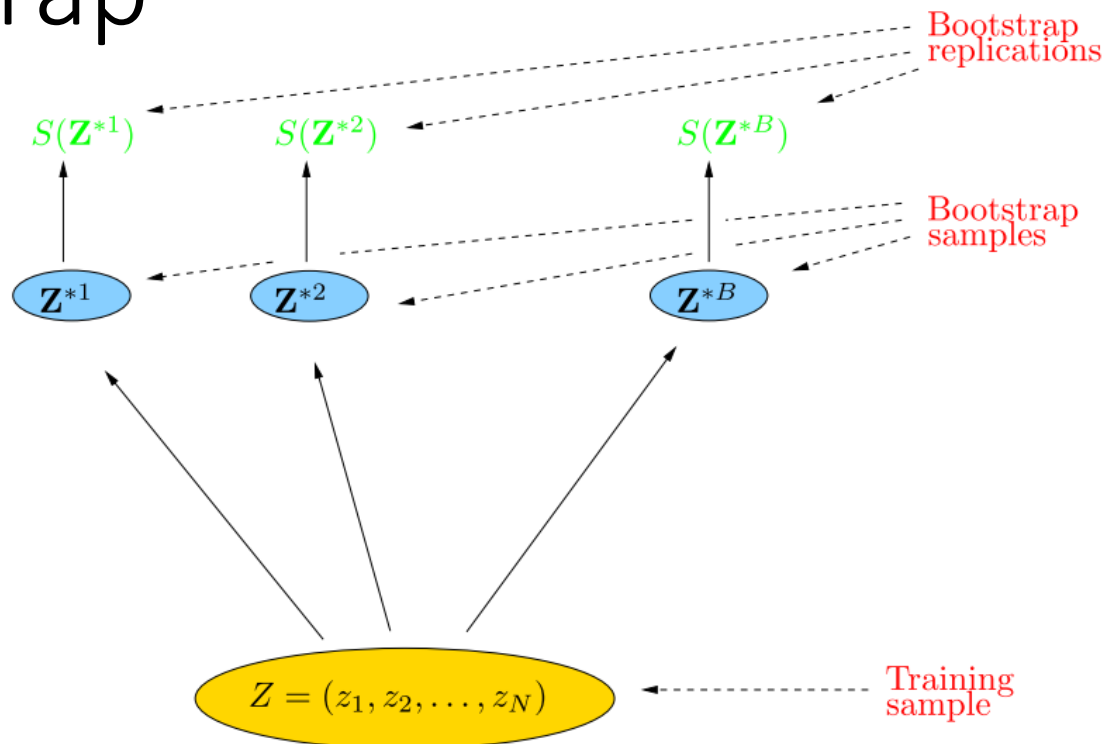
- Basic idea
  - Randomly draw datasets with replacement from the training data
  - Each replicate with the same size as the training set
  - Evaluate any statistics $S()$ over the replicates
    - For example, variance
      $$\hat{\mathrm{Var}}[S(\boldsymbol{Z})] = \frac{1}{B-1} \sum_{b=1}^{B} (S(\boldsymbol{Z}^{*b}) - \bar{S}^*)^2$$

Bootstrap

- Basic idea
  - Randomly draw datasets with replacement from the training data
  - Each replicate with the same size as the training set
  - Evaluate any statistics $S()$ over the replicates
    - For example, model error

$$\hat{\text{Err}}_{\text{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i))$$

Bootstrap for Model Evaluation

- If we directly evaluate the model using the whole training data

$$\hat{\mathrm{Err}}_{\mathrm{boot}} = \frac{1}{B} \frac{1}{N} \sum_{b=1}^{B} \sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i))$$

- As the probability of a data instance in the bootstrap samples is

$$P\{\text{observation } i \in \text{bootstrap samples}\} = 1 - \left(1 - \frac{1}{N}\right)^N$$

$$\simeq 1 - e^{-1} = 0.632$$

- If validate on training data, it is much likely to overfit
  - For example in a binary classification problem where *y* is indeed independent with *x*
    - Correct error rate: 0.5
    - Above bootstrap error rate: 0.632*0 + (1-0.632)*0.5=0.184

Leave-One-Out Bootstrap

- Build a bootstrap replicate with one instance *i* out, then evaluate the model using instance *i*

$$\hat{\text{Err}}^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(y_i, \hat{f}^{*b}(x_i))$$

  - C^{-i} is the set of indices of the bootstrap samples *b* that do not contain the instance *i*
  - For some instance *i*, the set C^{-i} could be null set, just ignore such cases

- We shall come back to the model evaluation and select in later lectures.

Bootstrap for Model Parameters

- Sec 8.4 of Hastie et al. The elements of statistical learning. 2008.

- Bootstrap mean is approximately a posterior average.

Bagging: Bootstrap Aggregating

- Bootstrap replication
  - Given n training samples $Z = \{(x_1,y_1), (x_2,y_2),...,(x_n,y_n)\}$, construct a new training set Z^* by sampling n instances with replacement
  - Construct B bootstrap samples Z^{*b} , $b = 1,2,...,B$
  - Train a set of predictors $\hat{f}^{*1}(x), \hat{f}^{*2}(x), \ldots, \hat{f}^{*B}(x)$
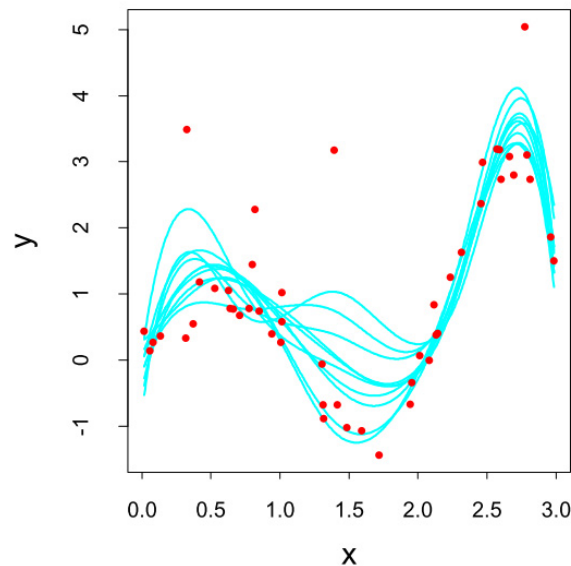
- Bagging average the predictions

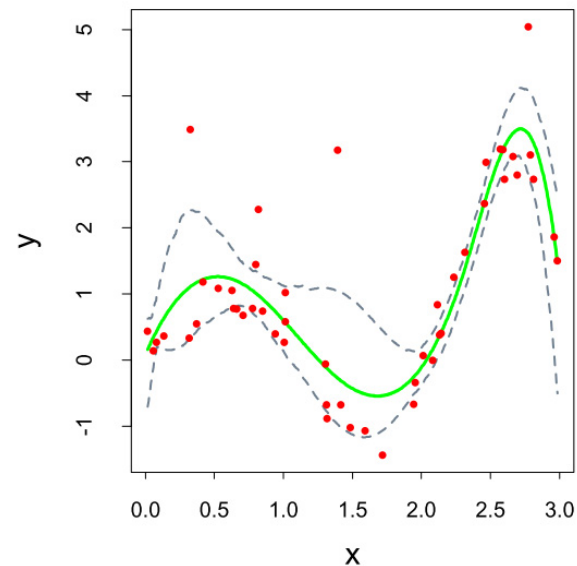$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$
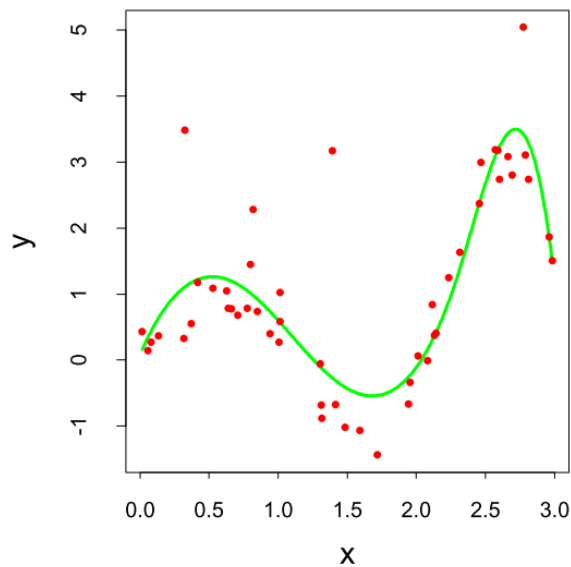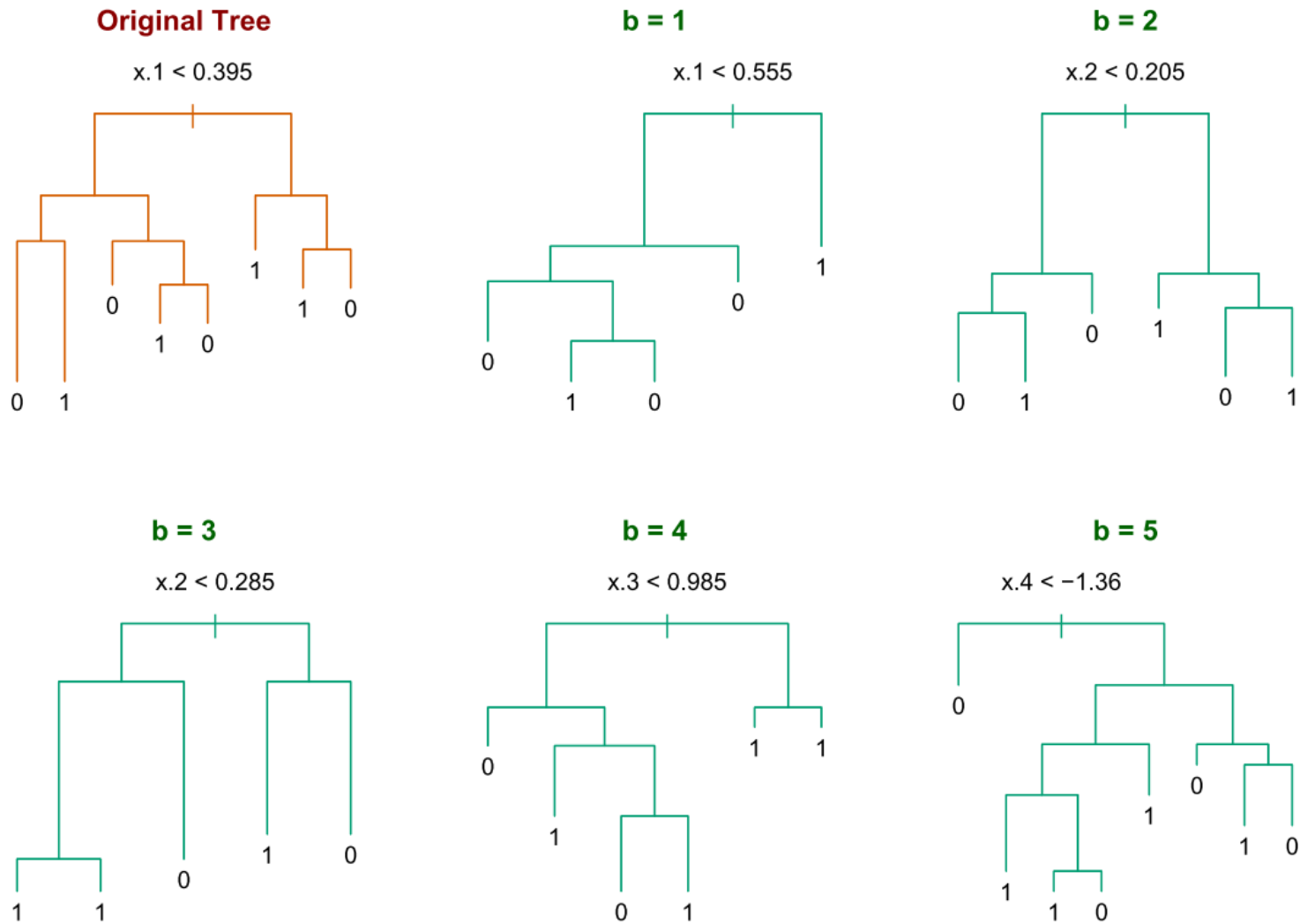
B-spline smooth of data
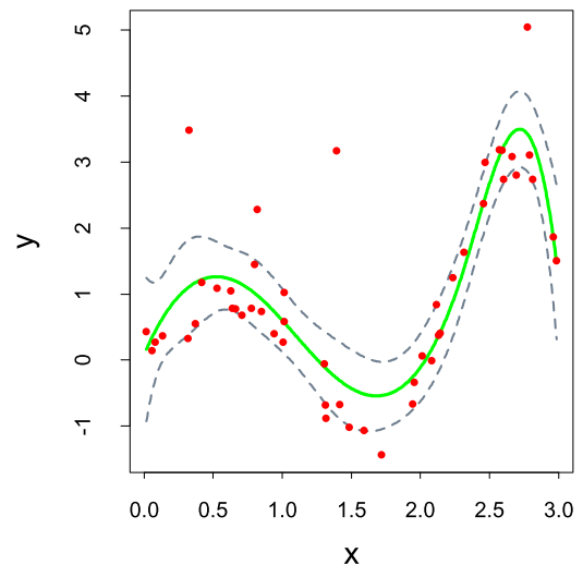
B-spline smooth plus and minus $1.96\times$ standard error bands

Ten bootstrap replicates of the B-spline smooth.

B-spline smooth with 95% standard error bands computed from the bootstrap distribution

Fig 8.2 of Hastie et al. The elements of statistical learning.

Bagging trees on simulated dataset. The top left panel shows the original tree. 5 trees grown on bootstrap samples are shown. For each tree, the top split is annotated.

Fig 8.9 of Hastie et al. The elements of statistical learning.
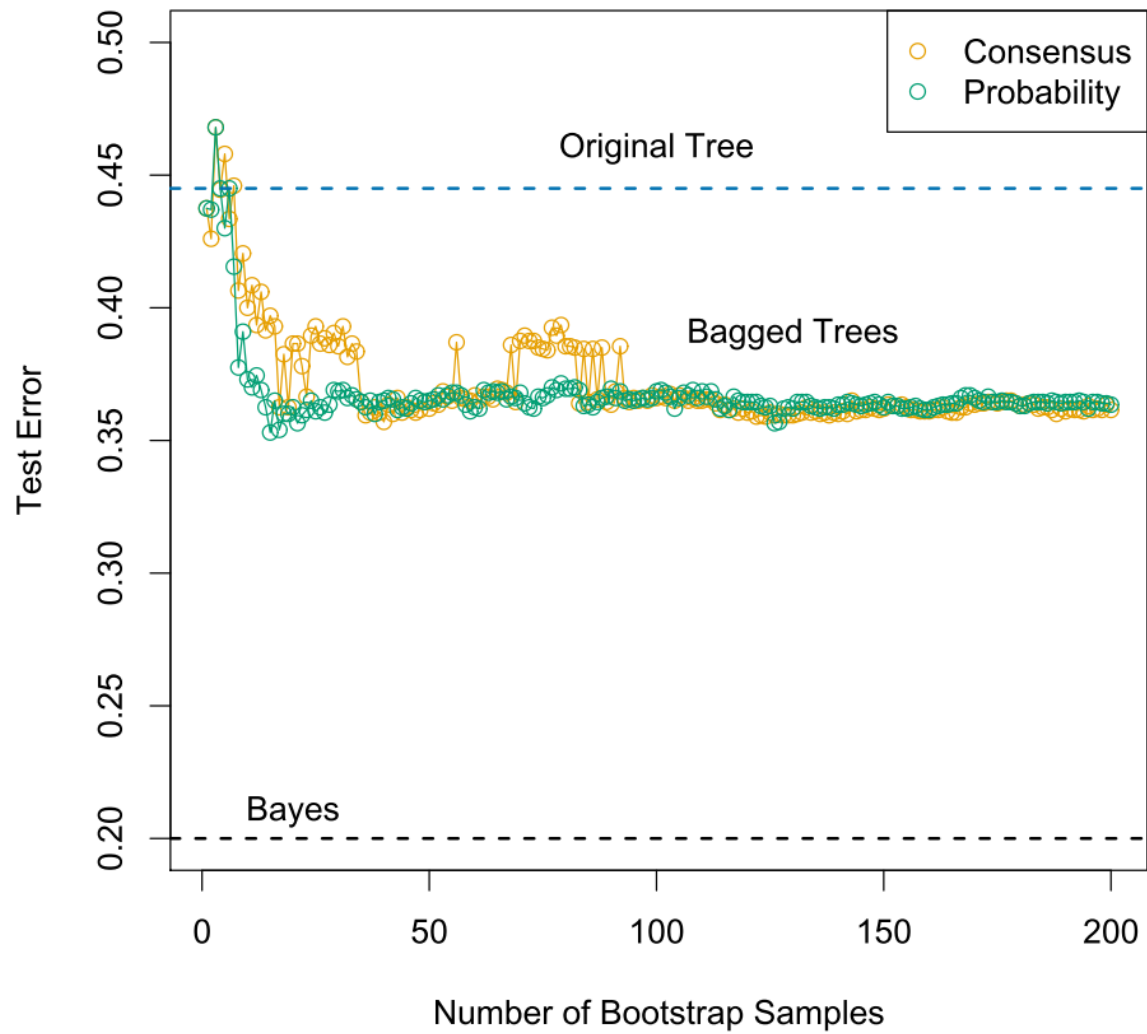
For classification bagging, consensus vote vs. class probability averaging

Fig 8.10 of Hastie et al. The elements of statistical learning.

Why Bagging Works

- Bias-Variance Decomposition
  - Assume $Y = f(X) + \epsilon$ where $\mathbb{E}[\epsilon] = 0$ $\mathrm{Var}[\epsilon] = \sigma_\epsilon^2$
  - Then the expected prediction error at an input point x_0

$$\mathrm{Err}(x_0) = \mathbb{E}[(Y - \hat{f}(x_0))^2 | X = x_0]$$
$$= \sigma_\epsilon^2 + [\mathbb{E}[\hat{f}(x_0)] - f(x_0)]^2 + \mathbb{E}[\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)]]^2$$
$$= \sigma_\epsilon^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0))$$

- Bagging works by reducing the variance with the same bias as the original model (trained over the whole data)
  - Works especially well based on low-bias and high-variance prediction models

Content of this lecture

- Ensemble Methods

- Bagging

- **Random Forest**

- AdaBoost

- Gradient Boosting Decision Trees

The Problem of Bagging

- <mark>Bagging works by reducing the variance</mark> with the same bias as the original model (trained over the whole data)
  - Works especially based on low-bias and high-variance prediction models

- If the variables (with variance σ^2) are i.d. (identically distributed but not necessarily independent) with positive correlation ρ, the variance of the average is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

  which reduces to $\rho\sigma^2$ if the bootstrap sample size goes to infinity

The Problem of Bagging

- Bagging works by reducing the variance with the same bias as the original model (trained over the whole data)

    - Works especially based on low-bias and high-variance prediction models

- Problem: the models trained from bootstrap samples are probably positively correlated

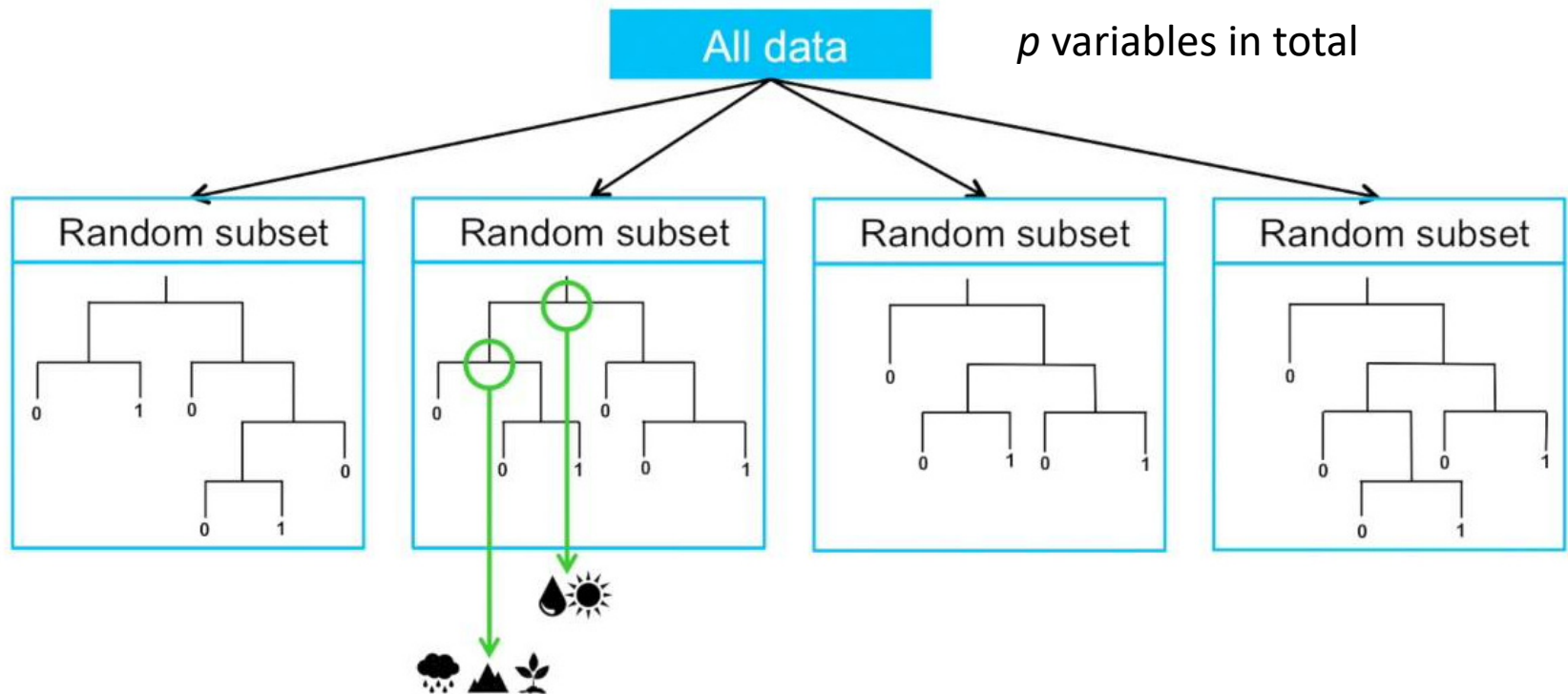$$\rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2$$

Random Forest

- Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): 532.
- Random forest is a substantial modification of bagging that builds a large collection of de-correlated trees, and then average them.

Image credit: https://i.ytimg.com/vi/-bYrLRMT3vY/maxresdefault.jpg
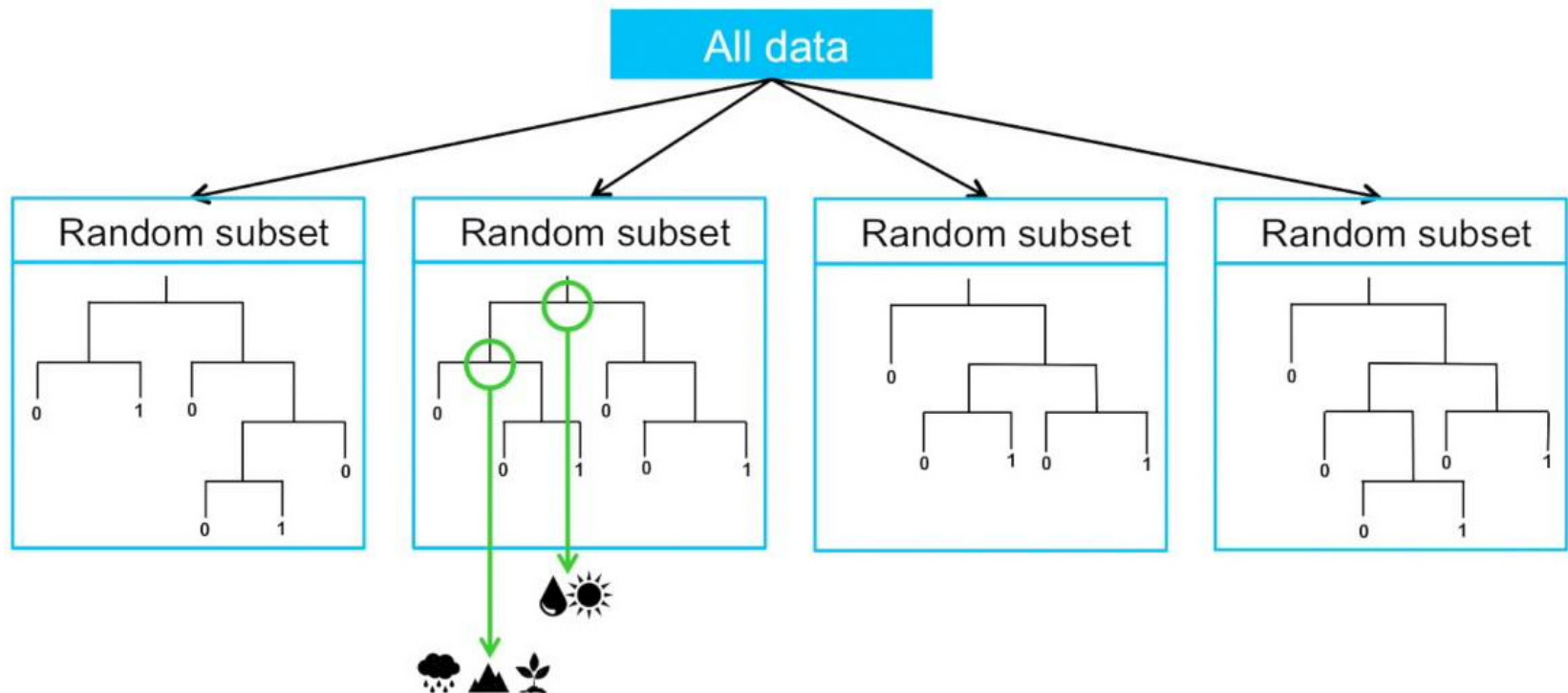
Tree De-correlation in Random Forest

- Before each tree node split, select *m* ≤ *p* variables at random as candidates of splitting
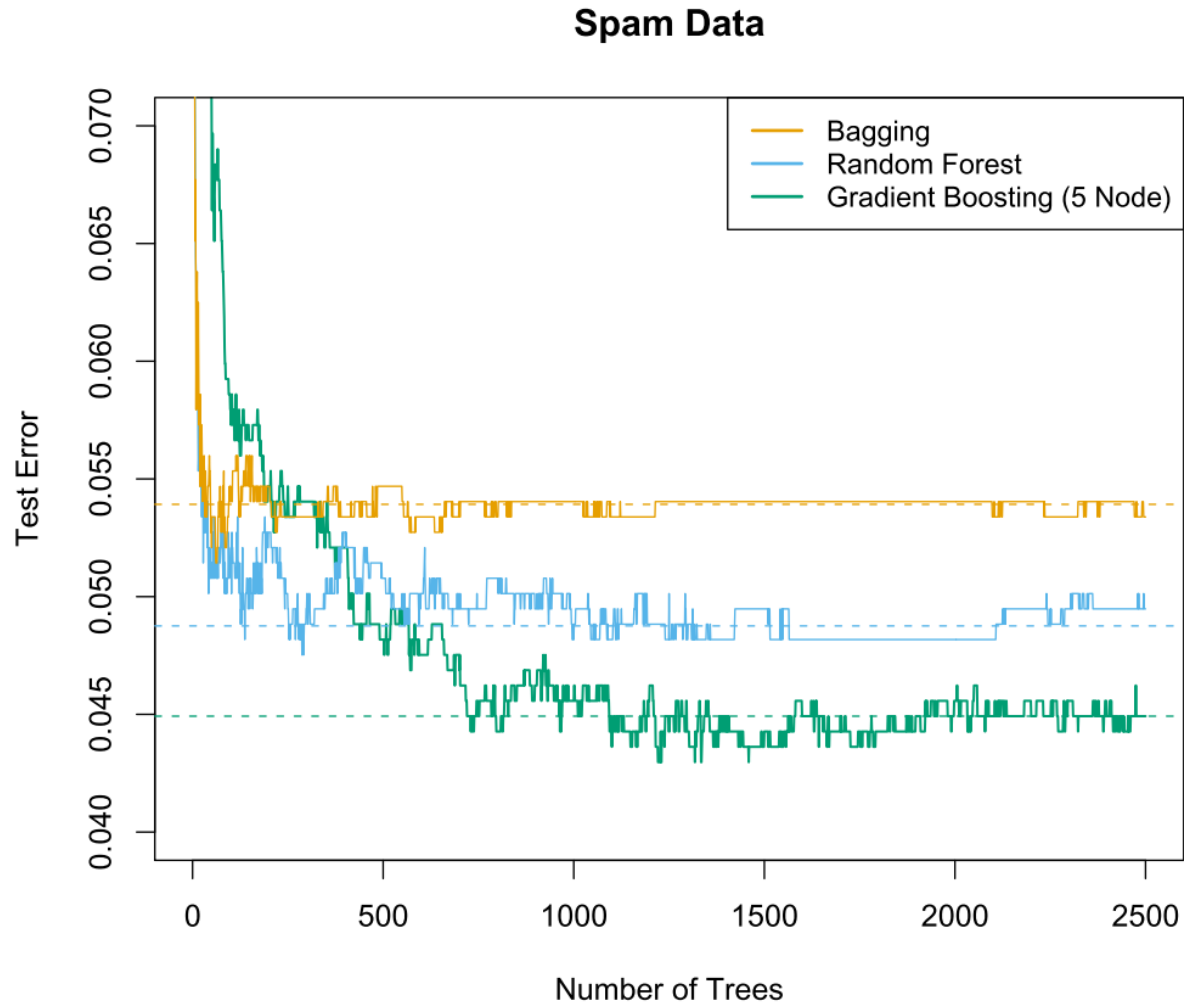 - Typically values $m = \sqrt{p}$ or even low as 1

Random Forest Algorithm

- For b = 1 to B:

    a) Draw a bootstrap sample $Z*$ of size n from training data

    b) Grow a random-forest tree T_b to the bootstrap data, by recursively repeating the following steps for each leaf node of the tree, until the minimum node size is reached

    I. Select m variables at random from the p variables

    II. Pick the best variable & split-point among the m

    III. Split the node into two child nodes

- Output the ensemble of trees $\{T_b\}_{b=1\ldots B}$

- To make a prediction at a new point x

    Regression: prediction average

    $$\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$$

    Classification: majority voting

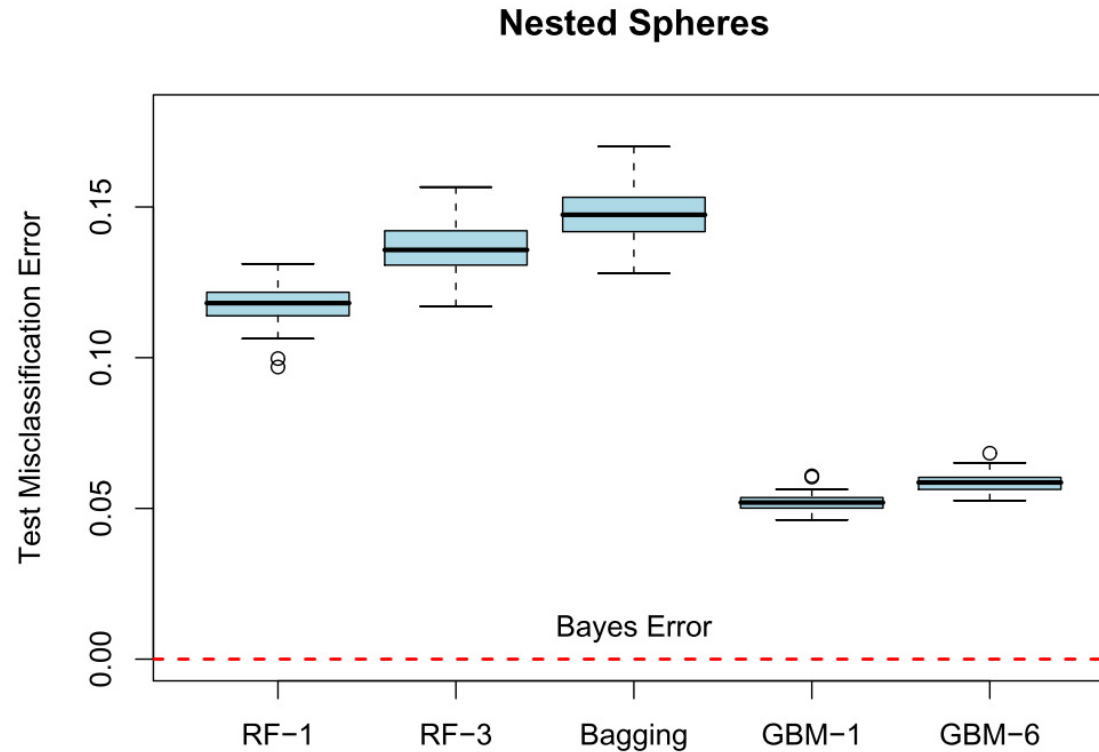 $$\hat{C}_{\mathrm{rf}}^B(x) = \mathrm{majority\ vote}\ \{\hat{C}_b(x)\}_1^B$$

Algorithm 15.1 of Hastie et al. The elements of statistical learning.

Performance Comparison

Spam Data

1536 test data instances

Performance Comparison

Nested Spheres

- Nest spheres data $Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > 9.34 \\ -1 & \text{otherwise} \end{cases}$

- RF-*m*: *m* means # of the randomly selected variables for each splitting

Fig. 15.2 of Hastie et al. The elements of statistical learning.

Content of this lecture

- Ensemble Methods

- Bagging

- Random Forest

- AdaBoost

- Gradient Boosting Decision Trees

Bagging vs. Random Forest vs. Boosting

- Bagging (bootstrap aggregating) simply treats each predictor trained on a bootstrap set with the same weight

- Random forest tries to de-correlate the bootstrap-trained predictors (decision trees) by sampling features

- Boosting strategically learns and combines the next predictor based on previous predictors

Additive Models and Boosting

- Strongly recommend:

  - Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. "Additive logistic regression: a statistical view of boosting." The annals of statistics 28.2 (2000): 337-407.

Additive Models

- General form of an additive model

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

- For regression problem

$$f_m(x) = \beta_m b(x; \gamma_m) \qquad F_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

  - Least-square learning of a predictor with others fixed $\quad$ argmin{F(X)}

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta,\gamma} \mathbb{E}\left[y - \sum_{k \neq m} \beta_k b(x; \gamma_k) - \beta b(x; \gamma)\right]^2$$

  - Stepwise least-square learning of a predictor with previous ones fixed

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta,\gamma} \mathbb{E}\left[y - F_{m-1}(x) - \beta b(x; \gamma)\right]^2$$ argmin{Fm-1(X)}

Additive Regression Models

- Least-square learning of a predictor with others fixed

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta, \gamma} \mathbb{E}\Big[y - \sum_{k \neq m} \beta_k b(x; \gamma_k) - \beta b(x; \gamma)\Big]^2$$

  - Essentially, the additive learning is equivalent with modifying the original data as

$$y_m \leftarrow y - \sum_{k \neq m} f_k(x)$$

- Stepwise least-square learning of a predictor with previous ones fixed

$$\{\beta_m, \gamma_m\} \leftarrow \arg\min_{\beta, \gamma} \mathbb{E}\Big[y - F_{m-1}(x) - \beta b(x; \gamma)\Big]^2$$

  - Essentially, the additive learning is equivalent with modifying the original data as

$$y_m \leftarrow y - F_{m-1}(x) = y_{m-1} - f_{m-1}(x)$$

Additive Classification Models

- For binary classification $\quad y = \{1, -1\}$

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

$$P(y = 1|x) = \frac{\exp(F(x))}{1 + \exp(F(x))}$$

$$P(y = -1|x) = \frac{1}{1 + \exp(F(x))}$$

- The monotone logit transformation

$$\log \frac{P(y = 1|x)}{1 - P(y = 1|x)} = F(x)$$

AdaBoost

- For binary classification, consider minimizing the criterion

$$J(F) = \mathbb{E}[e^{-yF(x)}]$$

[proposed by Schapire and Singer 1998 as an upper bound on misclassification error]

- It is (almost) equivalent with logistic cross entropy loss
  - for y=+1 and -1 label

$$
\begin{aligned}
\mathcal{L}(y, x) &= -\frac{1+y}{2} \log \frac{e^{F(x)}}{1 + e^{F(x)}} - \frac{1-y}{2} \log \frac{1}{1 + e^{F(x)}} \\
&= -\frac{1+y}{2}\left(F(x) - \log(1 + e^{F(x)})\right) + \frac{1-y}{2} \log(1 + e^{F(x)}) \\
&= -\frac{1+y}{2} F(x) + \log(1 + e^{F(x)}) \\
&= \log \frac{1 + e^{F(x)}}{e^{\frac{1+y}{2} F(x)}} = \begin{cases} \log(1 + e^{F(x)}) & \text{if } y = -1 \\ \log(1 + e^{-F(x)}) & \text{if } y = +1 \end{cases} = \log(1 + e^{-yF(x)})
\end{aligned}
$$

AdaBoost: an Exponential Criterion

- For binary classification, consider minimizing the criterion

$$J(F) = \mathbb{E}[e^{-yF(x)}]$$

- Solution

$$\mathbb{E}[e^{-yF(x)}] = \int \mathbb{E}[e^{-yF(x)}|x]p(x)dx$$

$$\mathbb{E}[e^{-yF(x)}|x] = P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)}$$

$$\frac{\partial \mathbb{E}[e^{-yF(x)}|x]}{\partial F(x)} = -P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)}$$

$$\frac{\partial \mathbb{E}[e^{-yF(x)}|x]}{\partial F(x)} = 0 \quad \Rightarrow \quad F(x) = \frac{1}{2}\log\frac{P(y=1|x)}{P(y=-1|x)}$$

AdaBoost: an Exponential Criterion

- Solution

$$\frac{\partial \mathbb{E}[e^{-yF(x)}|x]}{\partial F(x)} = 0 \quad \Rightarrow \quad F(x) = \frac{1}{2} \log \frac{P(y=1|x)}{P(y=-1|x)}$$

$$\Rightarrow \quad P(y=1|x) = \frac{e^{2F(x)}}{1+e^{2F(x)}}$$

- Hence, AdaBoost and LR are equivalent up to a factor 2

Discrete AdaBoost $\qquad f(x) = \pm 1$

- Criterion $\quad J(F) = \mathbb{E}[e^{-yF(x)}]$

- Current estimate $\quad F(x)$

- Seek an improved estimate $\quad F(x) + cf(x)$

- Taylor series

$$f(a+x) = f(a) + \frac{f'(a)}{1!}x + \frac{f''(a)}{2!}x^2 + \frac{f'''(a)}{3!}x^3 + \cdots$$

- With second-order Taylor series

$$J(F + cf) = \mathbb{E}[e^{-y(F(x)+cf(x))}]$$

$$\simeq \mathbb{E}[e^{-yF(x)}(1 - ycf(x) + c^2y^2f(x)^2/2)]$$

$$= \mathbb{E}[e^{-yF(x)}(1 - ycf(x) + c^2/2)]$$

Note that $\quad y^2 = 1 \quad f(x)^2 = 1$

Discrete AdaBoost $\quad f(x) = \pm 1$

- Criterion $\quad J(F) = \mathbb{E}[e^{-yF(x)}]$

$$J(F + cf) \simeq \mathbb{E}[e^{-yF(x)}(1 - ycf(x) + c^2/2)]$$

- Solve f with fixed c

$$f = \arg\min_f J(F + cf) = \arg\min_f \mathbb{E}[e^{-yF(x)}(1 - ycf(x) + c^2/2)]$$

$$= \arg\min_f \mathbb{E}_w[1 - ycf(x) + c^2/2 \mid x]$$

$$= \arg\max_f \mathbb{E}_w[yf(x) \mid x] \quad (\text{for } c > 0)$$

where the weighted conditional expectation

$$\mathbb{E}_w[yf(x) \mid x] = \frac{\mathbb{E}[e^{-yF(x)} yf(x)]}{\mathbb{E}[e^{-yF(x)}]}$$

The weight is the normalized error factor $e^{-yF(x)}$ on each data instance

Discrete AdaBoost $\quad f(x) = \pm 1$

- Solve f with fixed c

$$f = \arg\min_f J(F + cf) = \arg\max_f \mathbb{E}_w[yf(x)|x] \quad (\text{for } c > 0)$$

$$\mathbb{E}_w[yf(x)|x] = \frac{\mathbb{E}[e^{-yF(x)}yf(x)]}{\mathbb{E}[e^{-yF(x)}]} \qquad \text{Weighted expectation}$$

  - i.e., train an $f()$ with each training data instance weighted proportional to its previous error factor $e^{-yF(x)}$

- Solution

$$f(x) = \begin{cases} 1, & \text{if } \mathbb{E}_w(y|x) = P_w(y = 1|x) - P_w(y = -1|x) > 0 \\ -1, & \text{otherwise} \end{cases}$$

Discrete AdaBoost $f(x) = \pm 1$

- Criterion $J(F) = \mathbb{E}[e^{-yF(x)}]$

- Solve *c* with fixed *f*

$$c = \arg\min_c J(F + cf) = \arg\min_c \mathbb{E}_w[e^{-cyf(x)}]$$

$$\frac{\partial \mathbb{E}_w[e^{-cyf(x)}]}{\partial c} = \mathbb{E}_w[-e^{-cyf(x)}yf(x)]$$

$$= \mathbb{E}_w[P(y \neq f(x)) \cdot e^c + (1 - P(y \neq f(x))) \cdot (-e^{-c})]$$

$$= \text{err} \cdot e^c + (1 - \text{err}) \cdot (-e^{-c}) = 0$$

$$\Rightarrow c = \frac{1}{2}\log\frac{1 - \text{err}}{\text{err}} \qquad \boxed{\text{err} = \mathbb{E}_w[1_{[y \neq f(x)]}]}$$

The overall error rate of the weighted instances

Discrete AdaBoost $f(x) = \pm 1$

- Criterion $J(F) = \mathbb{E}[e^{-yF(x)}]$

- Solve *c* with fixed *f*

$$c = \frac{1}{2} \log \frac{1 - \text{err}}{\text{err}}$$

$$\text{err} = \mathbb{E}_w[1_{[y \neq f(x)]}]$$
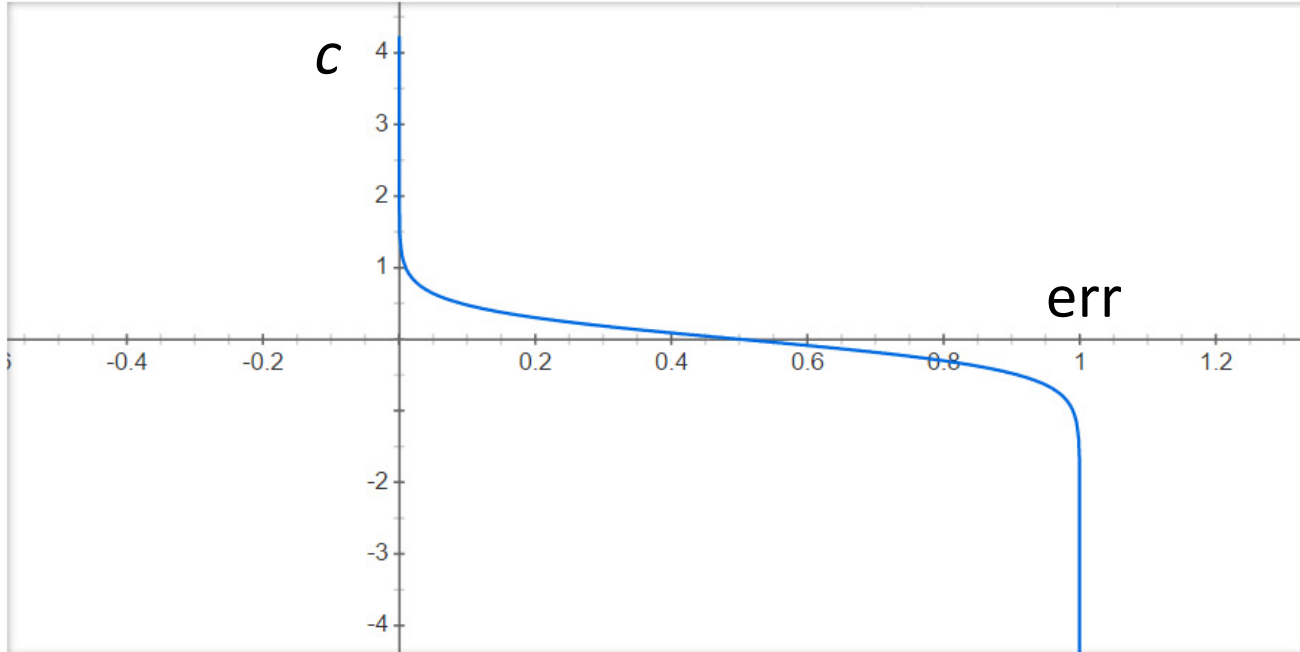
Discrete AdaBoost $f(x) = \pm 1$

- Iteration

  train *f*() with each training data instance weighted proportional to its error factor *e^{-yF(x)}*

$$f(x) = \begin{cases} 1, & \text{if } \mathbb{E}_w(y|x) = P_w(y = 1|x) - P_w(y = -1|x) > 0 \\ -1, & \text{otherwise} \end{cases}$$

$$\text{err} = \mathbb{E}_w[1_{[y \neq f(x)]}]$$

$$F(x) \leftarrow F(x) + \frac{1}{2} \log \frac{1 - \text{err}}{\text{err}} f(x)$$

$$w(x, y) \leftarrow w(x, y) e^{-cf(x)y} = w(x, y) e^{c(2 \times 1_{[y \neq f(x)]} - 1)}$$

$$= w(x, y) \exp \left(\log \frac{1 - \text{err}}{\text{err}} 1_{[y \neq f(x)]} - \frac{1}{2} \right)$$

Reduced after normalization

Discrete AdaBoost Algorithm

Discrete AdaBoost [Freund and Schapire (1996b)]

1. Start with weights $w_i = 1/N, i = 1, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Fit the classifier $f_m(x) \in \{-1, 1\}$ using weights w_i on the training data.

   (b) Compute $\mathrm{err}_m = E_w[1_{(y \neq f_m(x))}]$, $c_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.

   (c) Set $w_i \leftarrow w_i \exp[c_m 1_{(y_i \neq f_m(x_i))}]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.

3. Output the classifier $\mathrm{sign}[\sum_{m=1}^{M} c_m f_m(x)]$.

instance

—— c

Real AdaBoost Algorithm

Real AdaBoost

1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Fit the classifier to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$, using weights w_i on the training data.

   (b) Set $f_m(x) \leftarrow \frac{1}{2} \log p_m(x)/(1 - p_m(x)) \in R$.

   (c) Set $w_i \leftarrow w_i \exp[-y_i f_m(x_i)]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.

3. Output the classifier $\text{sign}[\sum_{m=1}^{M} f_m(x)]$.

- Real AdaBoost uses class probability estimates $p_m(x)$ to construct real-valued contributions $f_m(x)$.

$$\exp[-y\, fm(x)] \qquad\qquad < 1, \qquad\qquad > 1,$$

$$fm \qquad\qquad /$$
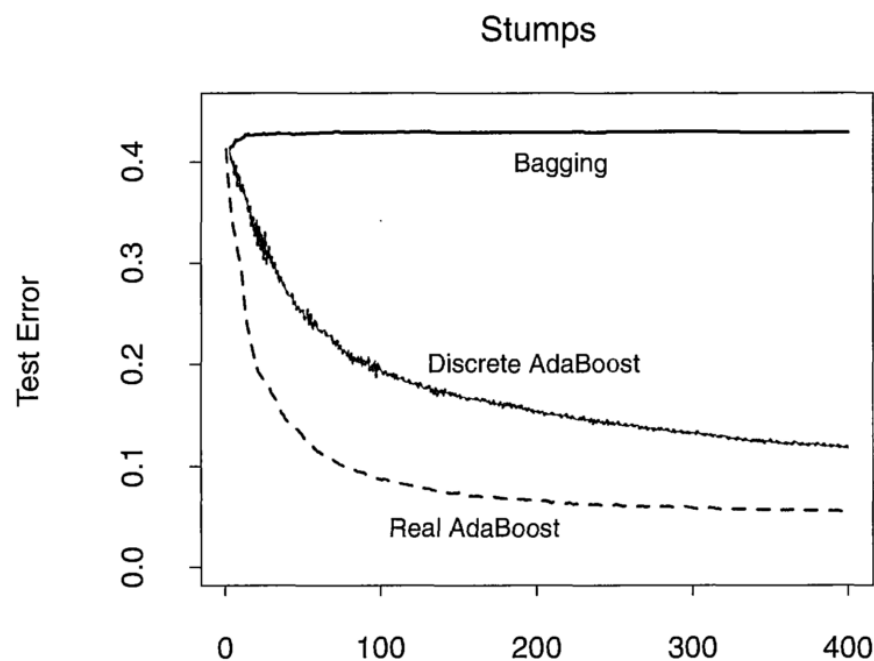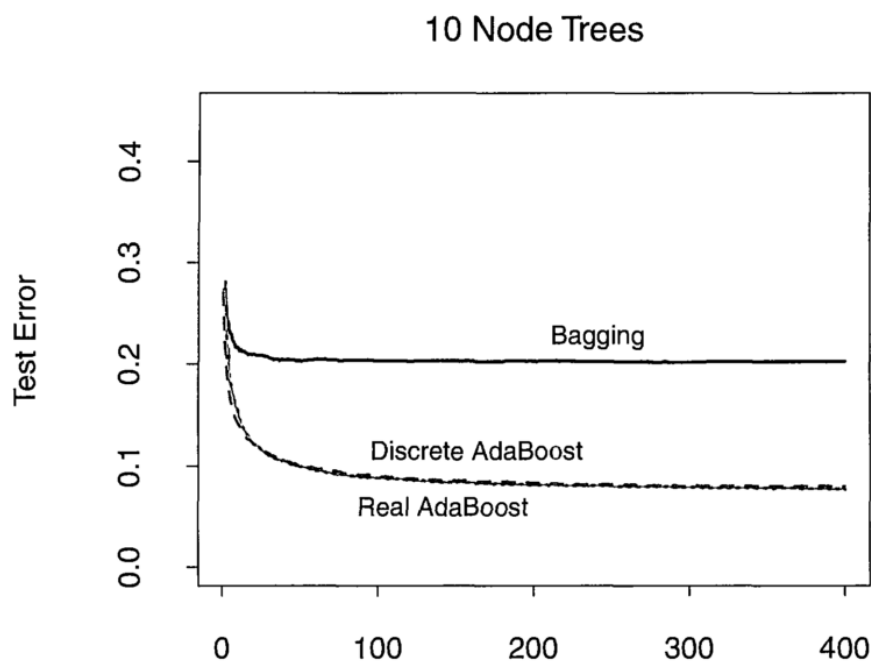
Bagging vs. Boosting

bagging

boosting

- Stump: a single-split tree with only two terminal nodes.

LogitBoost

LogitBoost (two classes)

1. Start with weights $w_i = 1/N$ $i = 1, 2, \ldots, N$, $F(x) = 0$ and probability estimates $p(x_i) = \frac{1}{2}$.
2. Repeat for $m = 1, 2, \ldots, M$:

   (a) Compute the working response and weights

   $$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))},$$
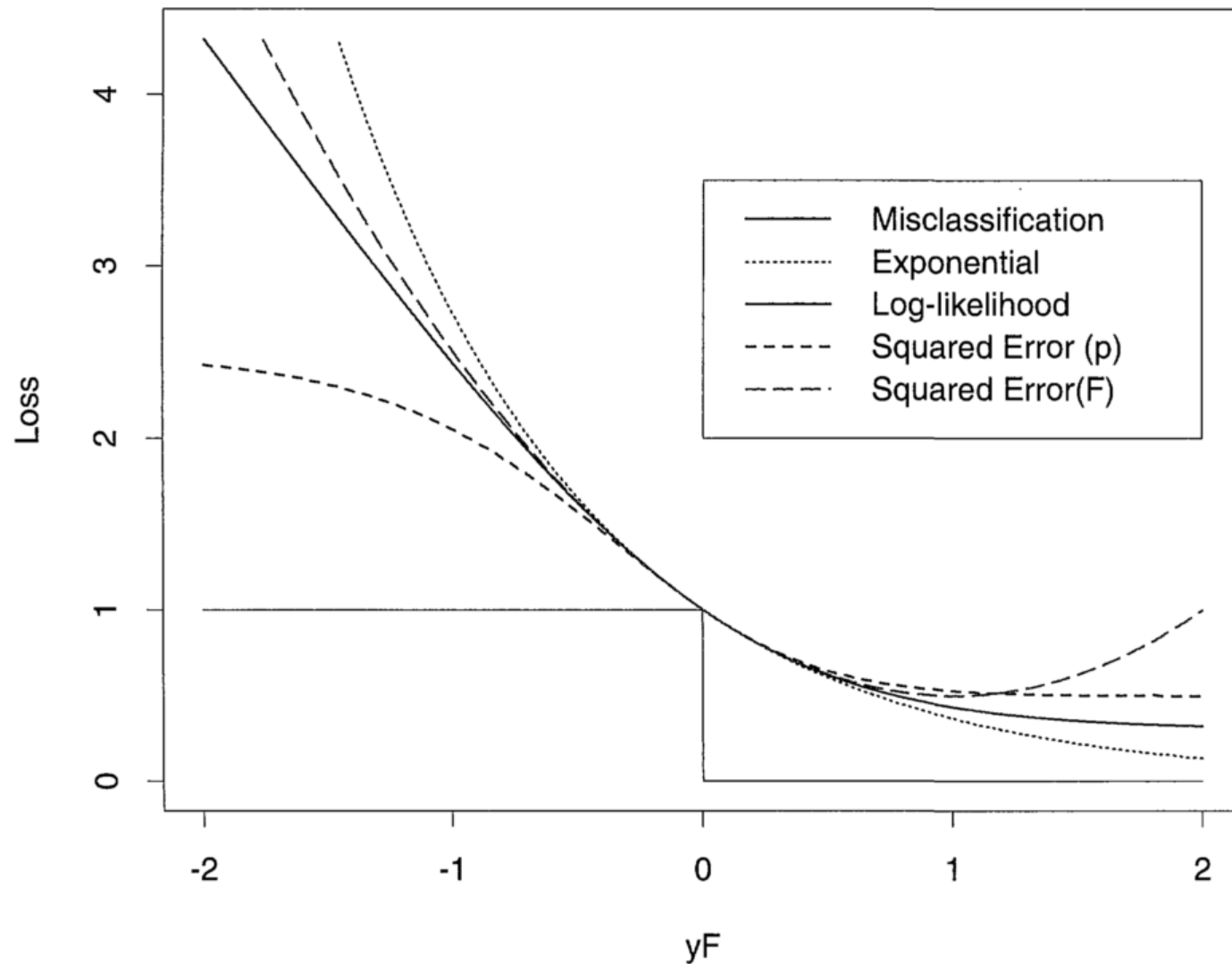
   $$w_i = p(x_i)(1 - p(x_i)).$$

   (b) Fit the function $f_m(x)$ by a weighted least-squares regression of z_i to x_i using weights w_i.

   (c) Update $F(x) \leftarrow F(x) + \frac{1}{2}f_m(x)$ and $p(x) \leftarrow (e^{F(x)})/(e^{F(x)} + e^{-F(x)})$.

3. Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$.

- More advanced than previous version of AdaBoost
- May not discussed in details

- Exponential criterion and log-likelihood (cross entropy) is equivalent on first 2 orders of Taylor series.

A Brief History of Boosting

- 1990 - Schapire showed that a weak learner could always improve its performance by training two additional classifiers on filtered versions of the input data stream
  - A weak learner is an algorithm for producing a two-class classifier with performance guaranteed (with high probability) to be significantly better than a coinflip

Robert Schapire

- Specifically
  - Classifier h_1 is learned on the original data with N samples
  - Classifier h_2 is then learned on a new set of N samples, half of which are misclassified by h_1
  - Classifier h_3 is then learned on N samples for which h_1 and h_2 disagree
  - The boosted classifier is h_B = Majority Vote(h_1, h_2, h_3)
  - It is proven h_B has improved performance over h_1

A Brief History of Boosting

- 1995 – Freund proposed a "boost by majority" variation which combined many weak learners simultaneously and improved the performance of Schapire's simple boosting algorithm
  - Both two algorithms require the weak learner has a fixed error rate

Yoav Freund

- 1996 – Freund and Schapire proposed AdaBoost
  - Dropped the fixed-error-rate requirement

- 1996~1998 – Freund, Schapire and Singer proposed some theory to support their algorithms, in the form of the upper bound of generalization error
  - But the bounds are too loose to be of practical importance
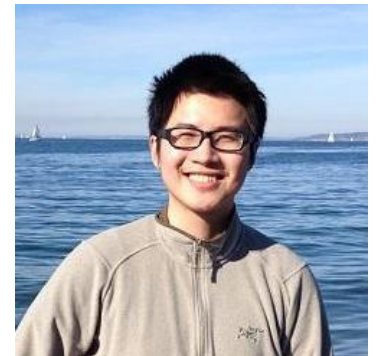 - Boosting achieves far more impressive performance than bounds

Content of this lecture

- Ensemble Methods

- Bagging

- Random Forest

- AdaBoost

- Gradient Boosting Decision Trees

Gradient Boosting Decision Trees

- Boosting with decision trees
  - $f_m(x)$ is a decision tree model
  - Many aliases such as GBRT, boosted trees, GBM

- Strongly recommend Tianqi Chen's tutorial
  - http://www.52cs.org/?p=429
  - http://homes.cs.washington.edu/~tqchen/data/pdf/BoostedTree.pdf
  - https://xgboost.readthedocs.io/en/latest/model.html

Additive Trees

$$J^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \Omega(f_t)$$

$$\hat{y}_i^{(t)} = \sum_{m=1}^{t} f_m(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Objective w.r.t. f_t $\quad J^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$

- Grow the next tree f_t to minimize the loss function $J^{(t)}$, including the tree penalty $\Omega(f_t)$

$$\min_{f_t} J^{(t)}$$

Taylor Series Approximation

Objective w.r.t. f_t
$$J^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

- Taylor series

$$f(a + x) = f(a) + \frac{f'(a)}{1!}x + \frac{f''(a)}{2!}x^2 + \frac{f'''(a)}{3!}x^3 + \cdots$$

- Let's define the gradients

$$g_i = \nabla_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \qquad h_i = \nabla_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$
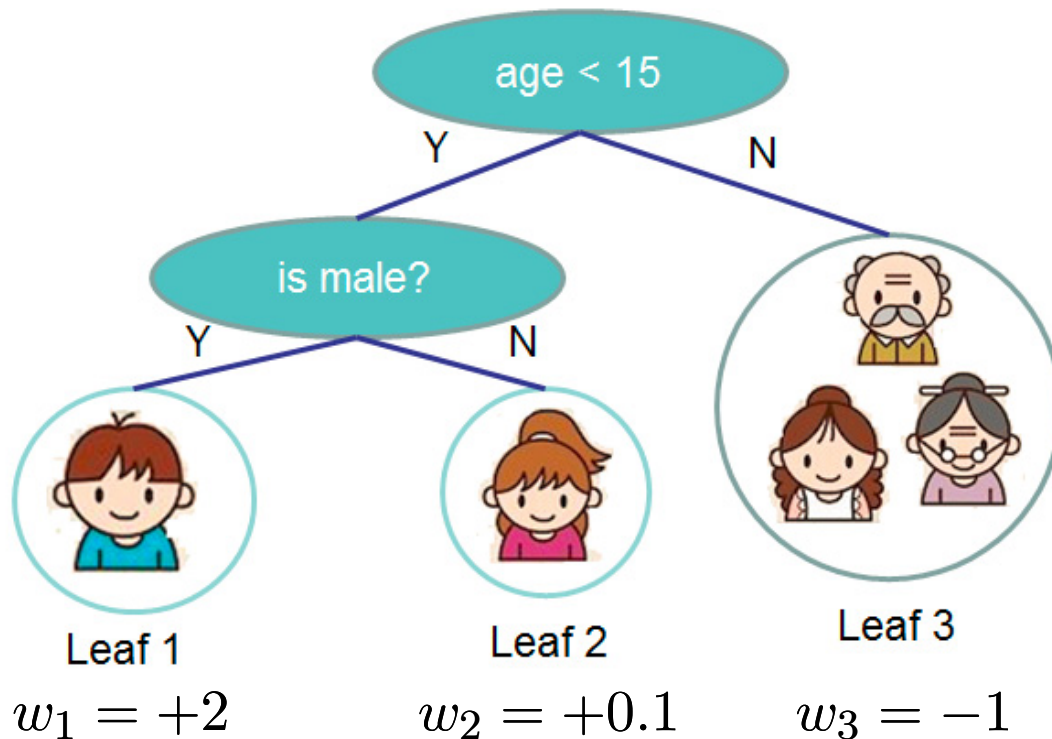
- Approximation

$$J^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

Penalty on Tree Complexity

- Prediction variety on leaves

$$f_t(x) = w_{q(x)}, \ w \in \mathbb{R}^T, \ q : \mathbb{R}^d \mapsto \{1, 2, \ldots, T\}$$
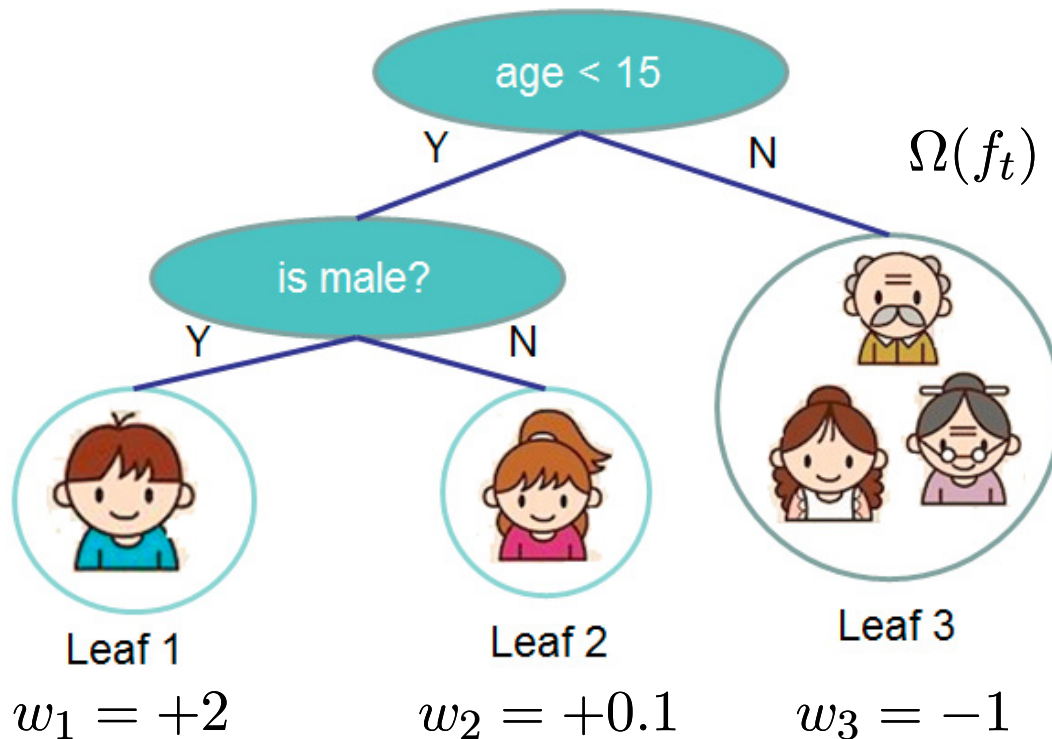
T: # leaves

Leaf 1
$w_1 = +2$

Leaf 2
$w_2 = +0.1$

Leaf 3
$w_3 = -1$

q( ) = 1

q( ) = 3

Penalty on Tree Complexity

- We could define the tree complexity as

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

size   weight

age < 15

Y   N

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

is male?

Y   N

Leaf 1   Leaf 2   Leaf 3

$w_1 = +2$ $w_2 = +0.1$ $w_3 = -1$

Rewritten Objective

- With the penalty $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

- Objective function

$$J^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

$$= \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \mathrm{const}$$

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

Sum over leaves

- I_j is the instance set $\{i \mid q(x_i) = j\}$

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

$$h_i = \nabla_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$

Rewritten Objective

- Objective function

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

- Define for simplicity $\quad G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

$$J^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

- With the fixed tree structure $\quad q : \mathbb{R}^d \mapsto \{1, 2, \ldots, T\}$

- The closed-form solution

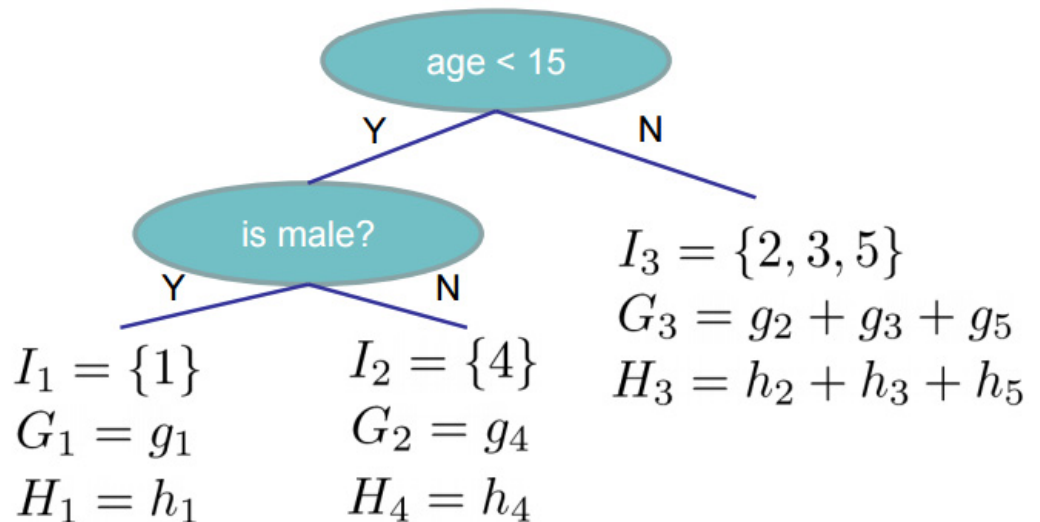$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

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

This measures how good a tree structure is

The Structure Score Calculation

Instance index   gradient statistics

1   g1, h1

2   g2, h2

3   g3, h3

4   g4, h4

5   g5, h5

age < 15

Y   N

is male?

Y   N

$I_1 = \{1\}$ $I_2 = \{4\}$
$G_1 = g_1$ $G_2 = g_4$
$H_1 = h_1$ $H_4 = h_4$

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

$$J^{(t)} = -\frac{1}{2} \sum_{j=1}^{3} \frac{G_j^2}{H_j + \lambda} + \gamma 3$$

The smaller, the better.

Reminder: this is already far from maximizing Gini impurity or information gain

Find the Optimal Tree Structure

- Feature and splitting point

- Greedily grow the tree
  - Start from 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

$$\text{Gain} = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

left child score   right child score   non-split score   penalty of the new leaf

Introducing a split may not obtain positive gain, because of the last term

Efficiently Find the Optimal Split

- For the selected feature *j*, we sort the data ascendingly

threshold **a**

g1, h1   g4, h4   g2, h2   g5, h5   g3, h3

$$G_L = g_1 + g_4 \qquad G_R = g_2 + g_3 + g_5$$

x_j
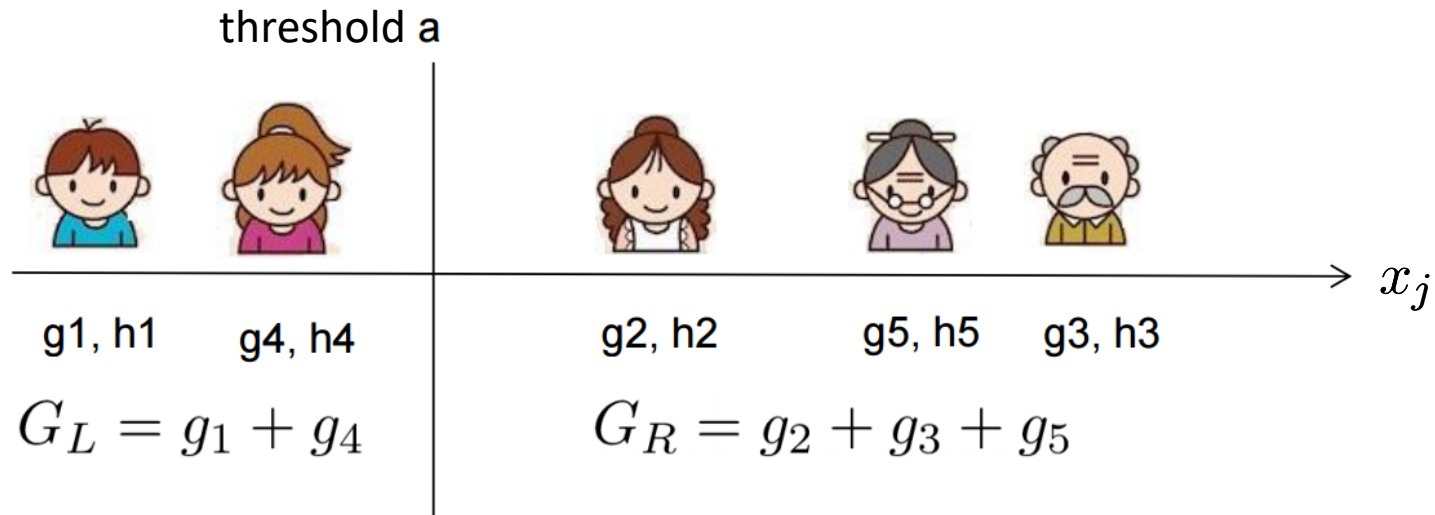
- All we need is sum of *g* and *h* on each side, and calculate

$$\text{Gain} = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

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

An Algorithm for Split Finding

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features

- Time Complexity growing a tree of depth K
  - It is $O(n\,d\,K \log n)$: or each level, need $O(n \log n)$ time to sort
    - There are d features, and we need to do it for K levels
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

XGBoost

- The most effective and efficient toolkit for GBDT

Scalable and Flexible Gradient Boosting

Star 11,352   Fork 5,168

Get Started

https://xgboost.readthedocs.io

T Chen, C Guestrin. XGBoost: A Scalable Tree Boosting System. KDD 2016.