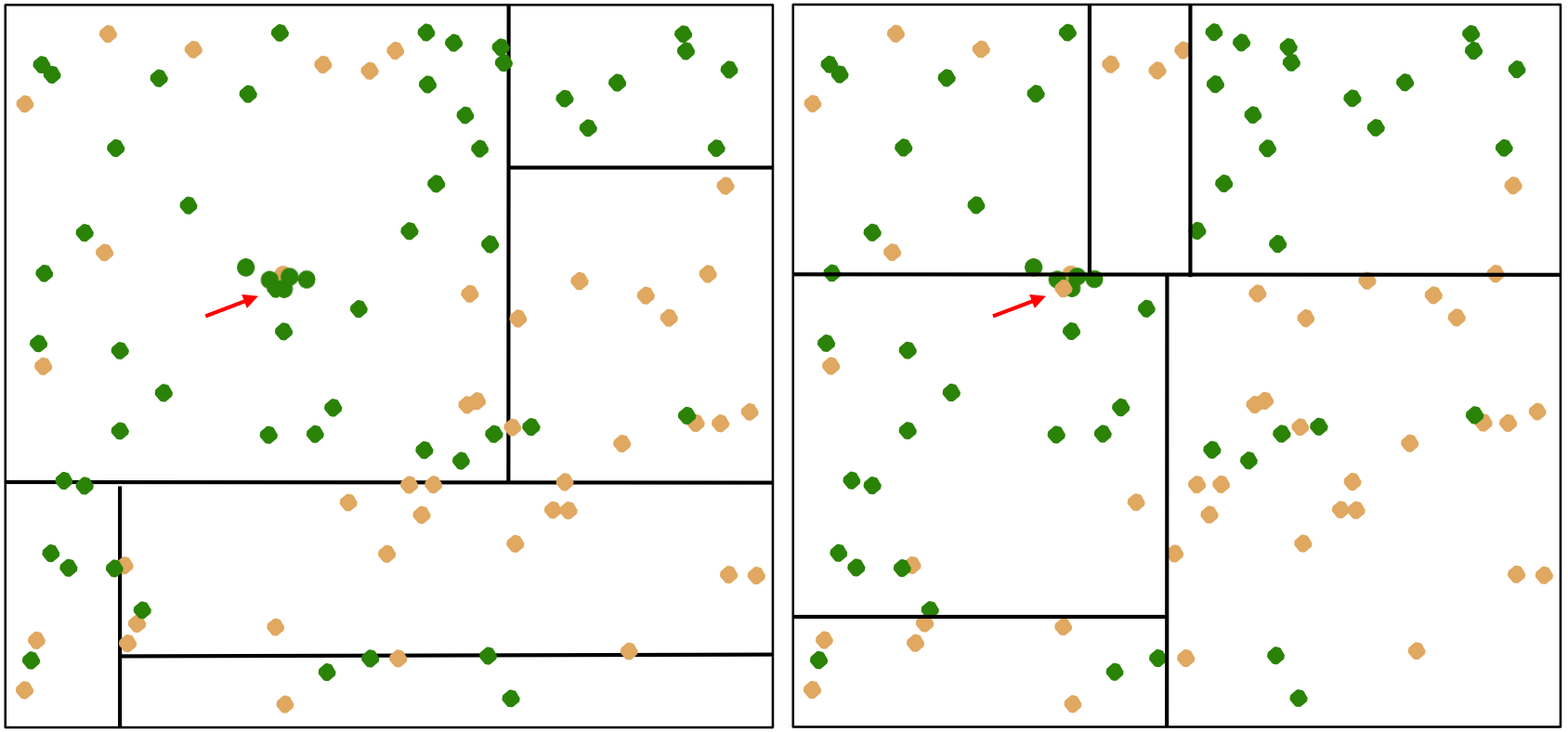
Chapter 5 Ensemble Methods

After completing this chapter, you are able to:

- Understand what ensemble methods are
- Perform bagging (bootstrap aggregating)
  - Perform random forest
- Perform boosting
  - Perform AdaBoost method
  - Perform Gradient Boost method

Trees are Unstable

One reversal



Accuracy = 81%

Accuracy = 80%

Fighting the Bias-Variance Tradeoff

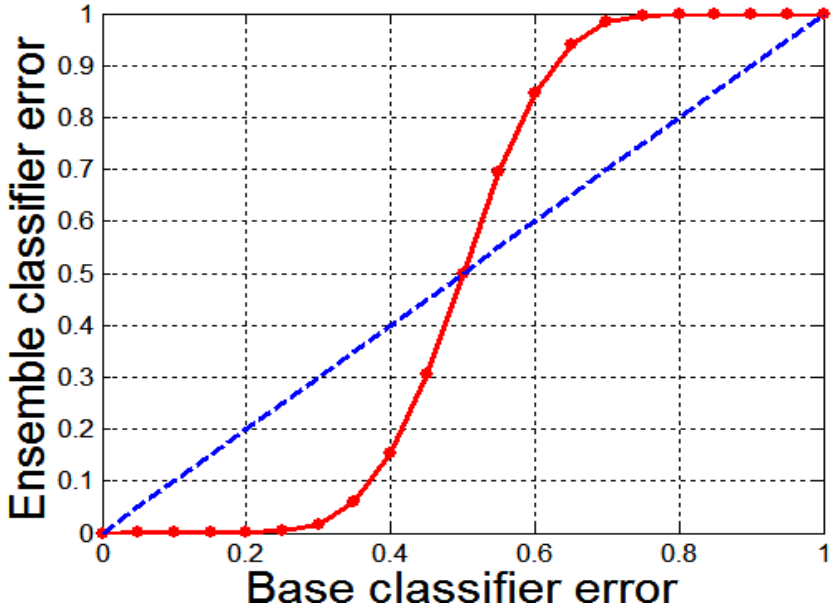
- Weak learners: poorly performed models
  - Hence cannot solve hard learning problems
- Some weak learners have low variance but high bias
  - e.g., linear/logistic regression
  - They usually do not overfit
- Some weak learners have large variance
  - e.g., decision tree (unstable prediction and hence large variance)
- Ensemble Methods aim to combine the predictions of several learners to improve generalizability / robustness (i.e., reduce variance) over a single learner.
- Two approaches:
  - Bagging; see, e.g., Random forest
  - Boosting ; see, e.g., Ada Boost, Gradient Boost

Ensemble Methods

- Instead of learning a single (weak or base) classifier, we construct many weak classifiers from the training data
- Predict the label/response of a test instance (or record) by combining the predictions made by these classifiers
  - Majority voting (class label only)
  - Averaging (class probabilities or predicted values)
  - Example: Consider 3 classifiers with probabilities of 0.4, 0.4 & 0.9 for Outcome 1. Then how to predict the label?
- YouTube: https://www.youtube.com/watch?v=Un9zObFjBH0&t=28s

Why Ensemble Methods work?

- Suppose there are 25 binary classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume errors made by classifiers are independent

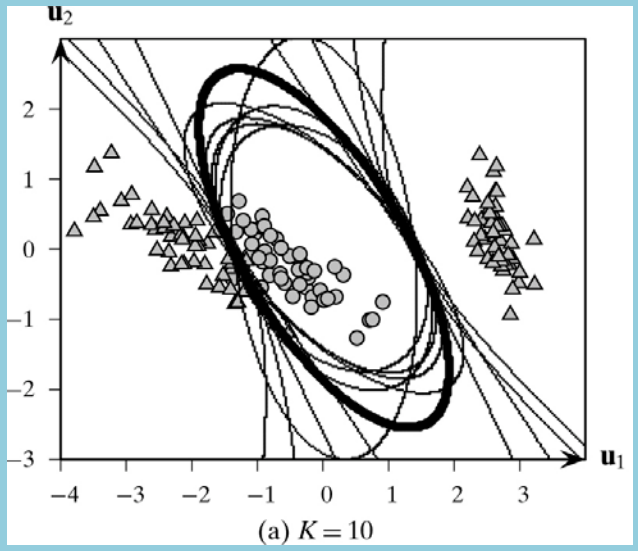


- Probability that the ensemble classifier (majority voting) makes a wrong prediction (no. of mistakes, X ≥ 13):

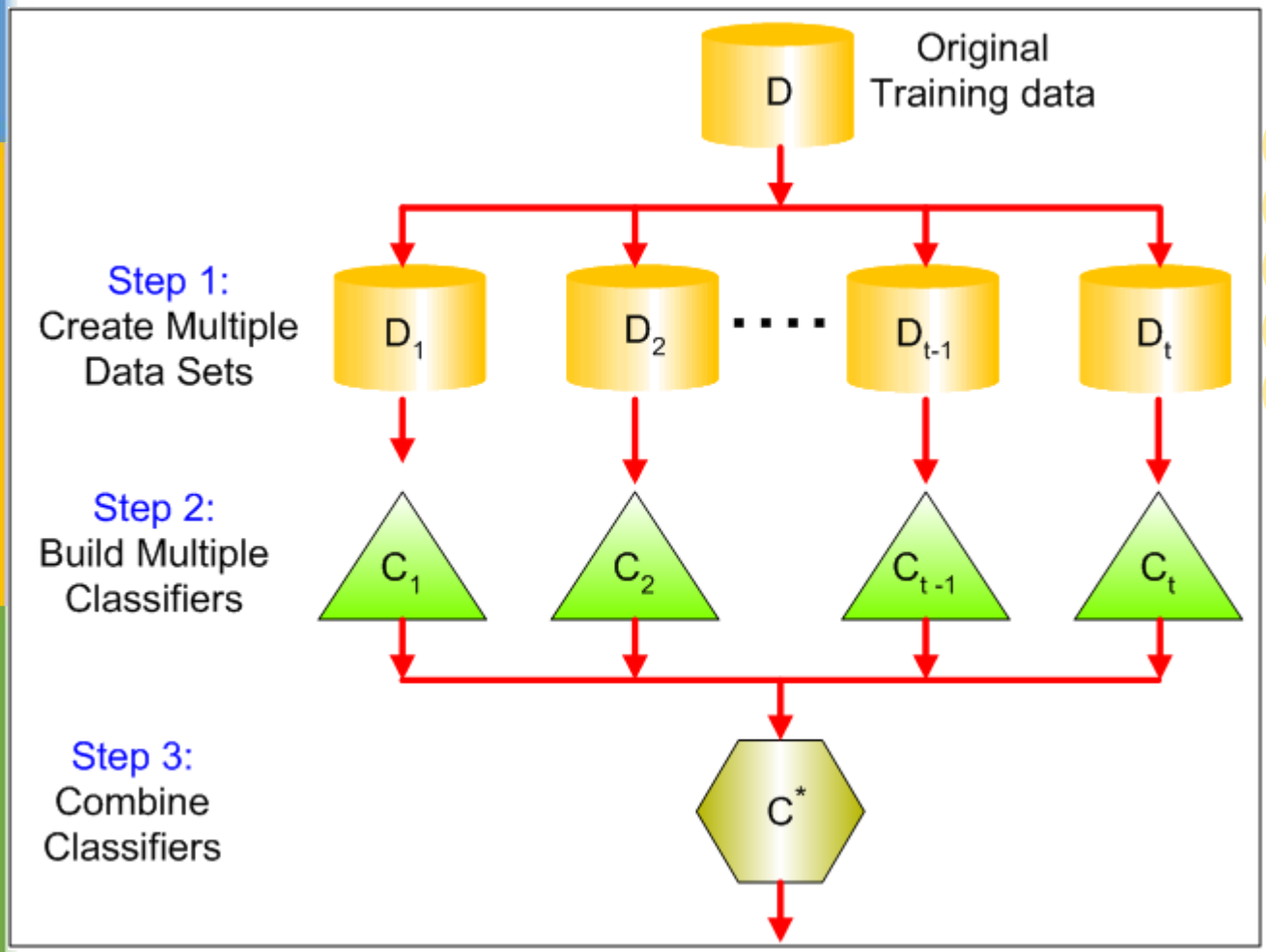
$$\mathrm{P}(\mathrm{X} \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i}$$

---三個臭皮匠頂個諸葛亮!!!

Bagging



Bagging and Pasting



Data sets can be dawn by bootstrapping (bagging) or random sampling without replacement (pasting)

To combine predictions, use majority voting for classification and the average for regression

We can use out-of-bag samples to estimate the generalization error

Bagging

- Also called bootstrap aggregating (Breiman 1996)
- Sampling with replacement from training data

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Each bootstrap sample has the same size as the original training data (can be different sample size)
- Build a base classifier on each bootstrap sample
- A test instance is assigned to the class that receives the highest number of votes

Refer to YouTube: https://www.youtube.com/watch?v=2Mg8QD0F1dQ

Bagging Algorithm

(1) Given a training set $\mathcal{D}$ of size N.

Let T be the number of bootstrap samples.

(2) For t=1 to T, repeat steps (a)-(b):

(a) From $\mathcal{D}$, create a bootstrap sample of size N

(b) Train a classifier C_t based on the bootstrap sample

(3) Classify a test instance x by

Voting: $C(x) = \underset{y}{\operatorname{argmax}} \sum_{t=1}^{T} \delta(C_t(x) = y)$

OR

Averaging: $C(x) = \underset{y}{\operatorname{argmax}} \frac{1}{T} \sum_{t=1}^{T} P(C_t(x) = y)$

Remarks on Bagging

- 63.2%
  - Suppose the size of original training data is N
  - Each data point has probability $1 – (1 – 1/N)^N$ of being selected in a bootstrap sample of size N
  - When $N \to \infty$, $1 – (1 – 1/N)^N \to 1 – 1/e = 0.632$
  - On average, a bootstrap sample contains approximately 63.2% of the original data
- For continuous response,
  - use average prediction

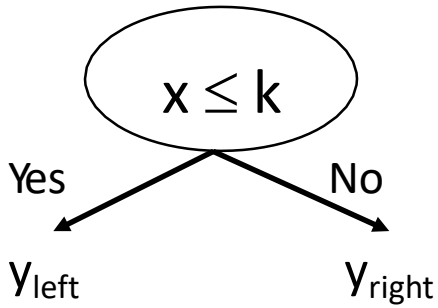
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump (tree with 1 split)
  - Decision rule: $\{x \leq k\}$ vs $\{x > k\}$
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 → y = 1
x > 0.7 → y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 → y = 1
x > 0.3 → y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 → y = 1
x > 0.05 → y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Bagging Performance (Classification Tree)

- Misclassification rates (Percent)

ML Dataset	err_{1tree}	err_{Bag}	Decrease
Waveform	29.0	19.4	33%
Heart	10.0	5.3	47%
Breast cancer	6.0	4.2	30%
Ionosphere	11.2	8.6	23%
Diabetes	23.4	18.8	20%
Glass	32.0	24.9	22%
Soybean	14.5	10.6	27%

50 bootstrap samples each

Bagging Performance (Regression Tree)

- Mean Squared Test set error

ML Dataset	err_{1tree}	err_{Bag}	Decrease
Boston	19.1	11.7	39%
Ozone	23.1	18.0	22%

50 bootstrap samples each

Random Forest

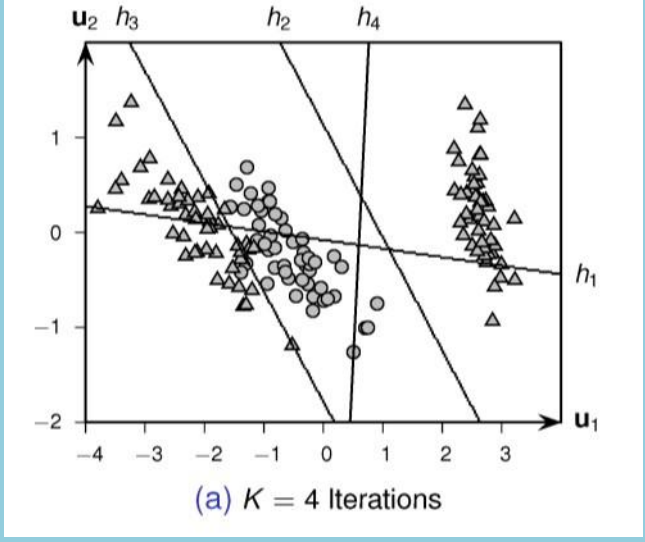
- What is random forest?
  YouTube: https://www.youtube.com/watch?v=J4Wdy0Wc_xQ
- In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
- When splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features.
- As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.
- In contrast to the original publication (Breiman,2001), the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.
- A feature's importance can be estimated by computing the average depth at which it appears across all trees in the forest.

Advantages and Disadvantages of Bagging

- Advantages
  - It can improve performance (accuracy) for unstable* learning algorithms (e.g. classification tree, neural network)
  - It can help reduce the variance, especially if the base classifiers are unstable, due to the averaging effect of majority voting. It does not, in general, have much effect on the bias
- Disadvantages
  - no simple, interpretable model
  - If the base classifier is stable, bagging may not improve the performance

* small changes in data lead to large changes in the prediction

Boosting



(a) $K = 4$ Iterations

Boosting

- An iterative procedure to adaptively change
  - the frequency weights of records in training data; OR
  - the training data by sampling based on the weights
- To *boost* the performance on previously misclassified records
  - Initially, assign equal weights to all N records
  - Unlike bagging, weights will be changed at the end of each boosting round
    - assign more weights to misclassified records and
    - assign less weights to correctly classified records

Discrete AdaBoost Algorithm

Adaptive Boosting (AdaBoost) (Freund & Schapire, 1996)

- Initialize the observation weights $w_i = \frac{1}{N}, i = 1, \ldots, N$
- For $t = 1, \ldots, T$, repeat steps (a)-(e):

  (a) Fit a classifier $C_t(x)$ to training data with weights $w_i, i = 1, \ldots, N$

  (b) Apply $C_t(x)$ to the original data to compute weighted error rate:

$$\varepsilon_t = \sum_{i=1}^{N} w_i I(C_t(x_i) \neq y_i),$$

  (c) If error rate $\varepsilon_t > 0.5$, set $w_i = \frac{1}{N}$ and go back to step (a)

  (d) Compute classifier's importance: $\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \geq 0$

  (e) Update observation weights: $w_i \leftarrow w_i \exp[\alpha_t I(C_t(x_i) \neq y_i)]$ and renormalize w_i to sum to 1

- Given $y \in \{-1,1\}$. After training the T classifiers, the combined classifier is:

$$C(x) = sign \sum_{t=1}^{T} \alpha_t C_t(x)$$

YouTube: https://www.youtube.com/watch?v=GM3CDQfQ4sw&t=20s

Remarks on AdaBoost

- In step (a), how to fit a classifier $C_t(x)$ to training data with different weights w_i's? AdaBoost appears in 2 versions:
  - Resampling – data is sampled with replacement based on weights
  - Reweighting – classifier takes into account the weights of obs. (eg. weighted least squares)
  - No strong evidence favoring one over the other

- Two equivalent definitions of weights:
  - In Freund & Schapire (1996) [AdaBoost.M1 (discrete AdaBoost)],

$$\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \qquad w_i \leftarrow w_i \exp\left[\alpha_t \delta\left(C_t(x_i) \neq y_i\right)\right]$$

  - In Breiman (1996),

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \qquad w_i \leftarrow \begin{cases} w_i \exp(\alpha_t) & \text{if } C_t(x_i) \neq y_i \\ w_i \exp(-\alpha_t) & \text{if } C_t(x_i) = y_i \end{cases}$$

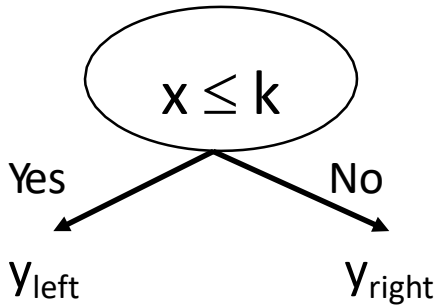
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $\{x \leq k\}$ vs $\{x > k\}$
  - Split point k is chosen based on entropy



AdaBoost Example

- Simulated training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Apply the fitted tree to original data to compute weighted error

Round	Split point	Left class	Right class	Weighted Error	alpha
1	0.75	-1	1	30%	0.8473
2	0.05	1	1	28.57%	0.9163
3	0.3	1	-1	15%	1.7346

AdaBoost Example

- Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.167	0.167	0.167	0.071	0.071	0.071	0.071	0.071	0.071	0.071
3	0.117	0.117	0.117	0.125	0.125	0.125	0.125	0.05	0.05	0.05

- Classification

=-1*0.84+1*0.92+1*1.73

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	1.80	1.80	1.80	-1.67	-1.67	-1.67	-1.67	0.029	0.029	0.029
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Discrete AdaBoost(Freund & Schapire 1996*b*)

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 $\text{err}_m = E_w[1_{(y \neq f_m(x))}]$, $c_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (c) Set $w_i \leftarrow w_i \exp[c_m \cdot 1_{(y_i \neq f_m(x_i))}]$, $i = 1, 2, \ldots N$, and renormalize so that $\sum_i w_i = 1$.
3. Output the classifier $\text{sign}[\sum_{m=1}^{M} c_m f_m(x)]$

For binary response

Algorithm 1: *E_w represents expectation over the training data with weights $w = (w_1, w_2, \ldots w_n)$, and $1_{(S)}$ is the indicator of the set S. At each iteration AdaBoost increases the weights of the observations misclassified by $f_m(x)$ by a factor that depends on the weighted training error.*

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 $f_m(x) \leftarrow \frac{1}{2} \log \frac{p_m(x)}{1 - p_m(x)} \in R$.
   (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 $\text{sign}[\sum_{m=1}^{M} f_m(x)]$

For binary response

Algorithm 2: *The Real AdaBoost algorithm uses class probability estimates $p_m(x)$ to construct real-valued contributions $f_m(x)$.*

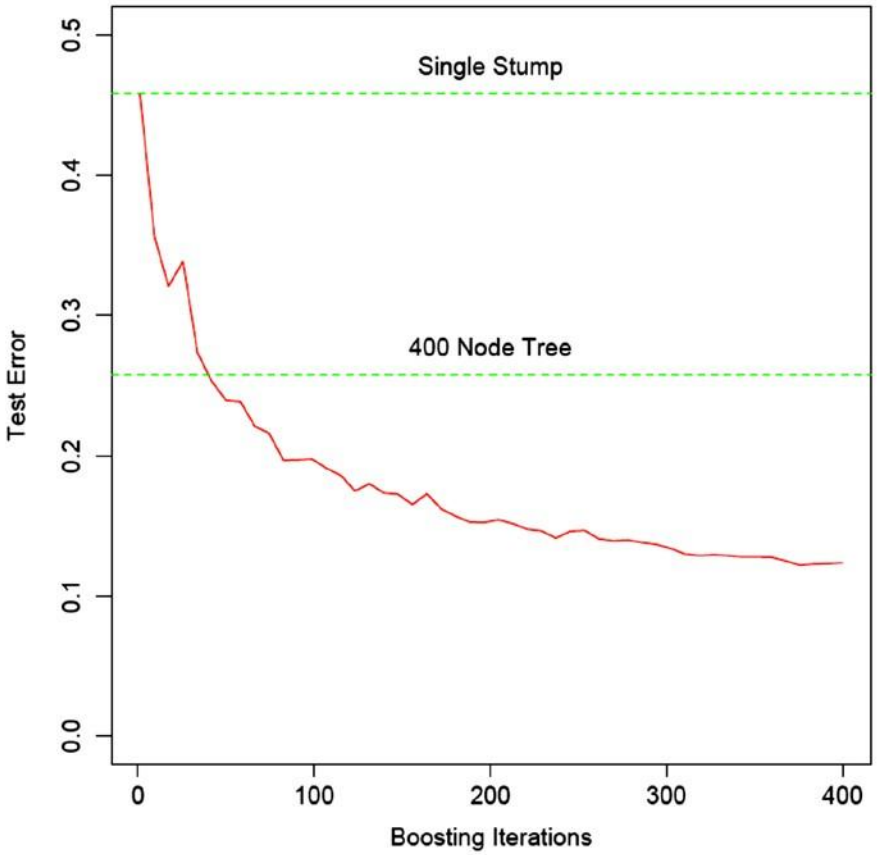
Boosting with Decision Stumps

- 2000 obs. simulated from

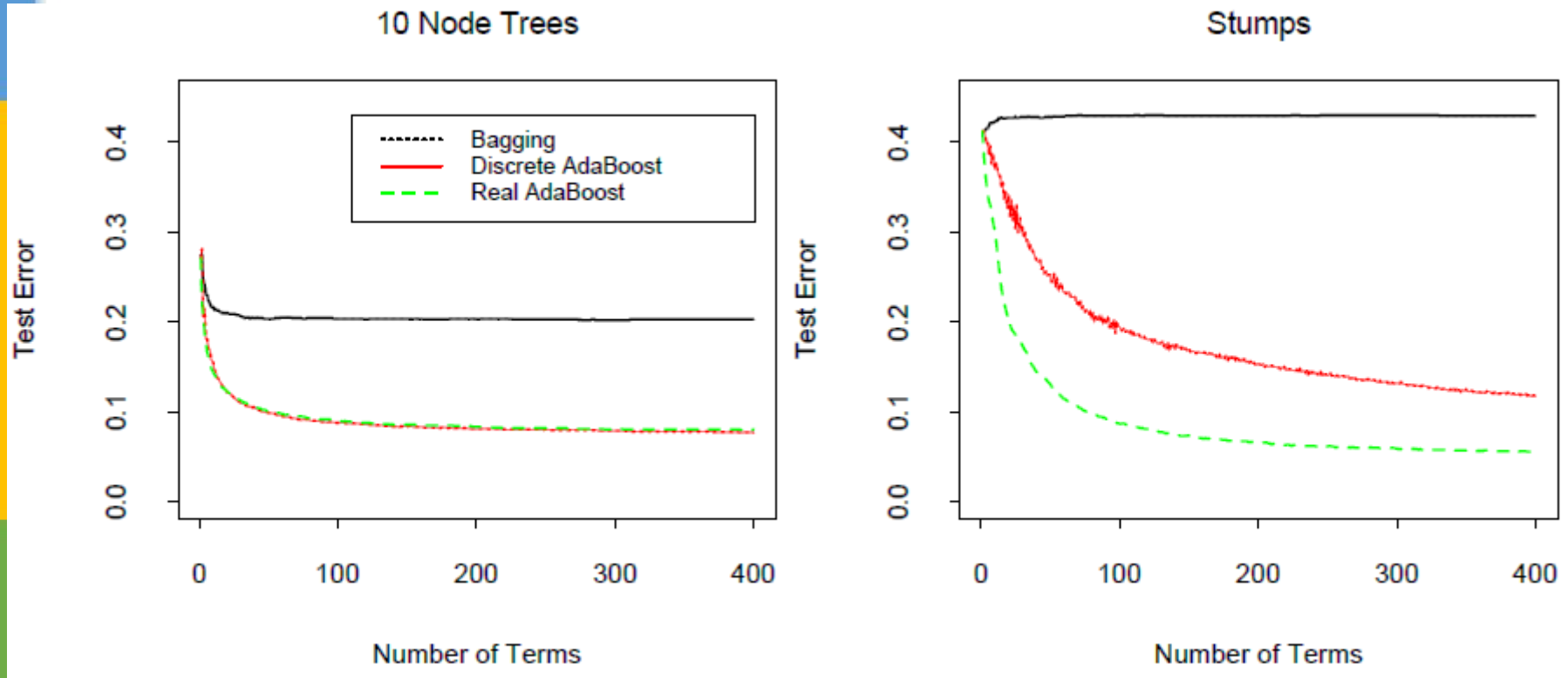
$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5) = 9.34 \\ -1 & \text{otherwise} \end{cases}$$

  the X_j's are iid N(0,1)

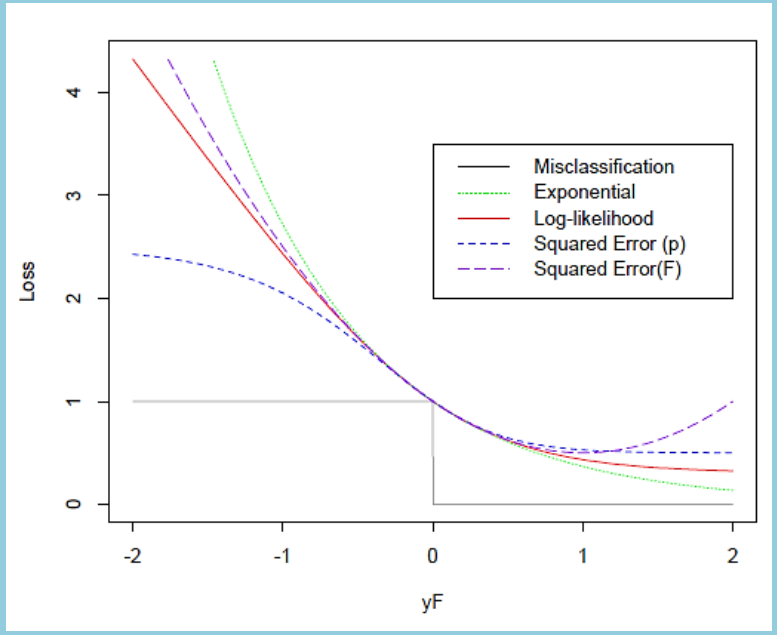
- A stump is a two-node tree after a single split.
  - random guessing: 50%
  - a single stump: 46%
  - a classification tree with 400 nodes: 26%
  - boosting with stump after 400 iterations: 12.6%



Bagging, Discrete AdaBoost and Real AdaBoost



Boosting as Forward Stagewise Additive Modeling



Boosting Fits an Additive Model

- Boosting builds an additive model

$$f_T(x) = \sum_{t=1}^{T} \beta_t b(x; \gamma_t)$$

  Where $b(x; \gamma_t)$ is a model with parameters γ_t (e.g. a tree with the splits depending on γ_t)
- Many of the learning techniques are of this form.
  - E.g. Linear regression, GAM, Basis expansion
- Traditionally the parameters (β_t, γ_t) are fit jointly (e.g. least squares, maximum likelihood)
- With boosting, the parameters (β_t, γ_t) are fit in a stagewise fashion. This slows the process down, and tends to overfit less quickly.

Forward Stagewise Additive Modeling

- Sequentially add new basis without adjusting those parameters that have already been added.
- Given a loss function L, e.g. $L\big(y, f_T(x)\big) = [y - f_T(x)]^2$
- Note that $f_T(x) = \sum_{t=1}^{T} \beta_t b(x; \gamma_t) = f_{T-1}(x) + \beta_T b(x; \gamma_T)$
- Algorithm : Forward stagewise additive modeling

1. Initialize $f_0 = 0$

2. For $t = 1$ to T

Compute $(\beta_t, \gamma_t) = \underset{\beta_t, \gamma_t}{\operatorname{argmin}} \sum_{i=1}^{N} L\big(y_i, f_t(x_i)\big)$

$$= \underset{\beta_t, \gamma_t}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i) + \beta_t b(x_i; \gamma_t))$$

Exponential Loss and AdaBoost

- Discrete AdaBoost is equivalent to forward stagewise additive modeling using exponential loss function

Exponential loss function $L\big(y, f_T(x)\big) = \exp[-y f_T(x)]\,, y \in \{1, -1\}$

with $f_T(x) = \sum_{t=1}^{T} \beta_t b(x; \gamma_t)$ and $b(x; \gamma_t) = C_t(x_i)$, a weak classifier

$$(\beta_t, C_t) = \underset{\beta_t, C_t}{\operatorname{argmin}} \sum_{i=1}^{N} \exp\big[-y_i\big(f_{t-1}(x_i) + \beta_t C_t(x_i)\big)\big]$$

$$= \underset{\beta_t, C_t}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(t)} \exp[-\beta_t y_i C_t(x_i)] \text{ where } w_i^{(t)} = \exp[-y_i f_{t-1}(x_i)]$$

$$\Rightarrow C_t = \underset{C_t}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(t)} \delta(C_t(x_i) \neq y_i) \text{ for } \beta_t > 0$$

- Hence,

$$\beta_t = \underset{\beta_t}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(t)} \exp[-\beta_t y_i C_t(x_i)]$$

$$= \underset{\beta_t}{\operatorname{argmin}} \left[e^{\beta_t} \sum_{y_i \neq C_t(x_i)} w_i^{(t)} + e^{-\beta_t} \sum_{y_i = C_t(x_i)} w_i^{(t)} \right]$$

$$= \underset{\beta_t}{\operatorname{argmin}} \left[e^{-\beta_t} \sum_{i=1}^{N} w_i^{(t)} + \left(e^{\beta_t} - e^{-\beta_t}\right) \sum_{i=1}^{N} w_i^{(t)} \delta(C_t(x_i) \neq y_i) \right]$$

- Solving for β_t, we have

$$\beta_t = \frac{1}{2} \ln \frac{1 - err_t}{err_t} \text{ where } err_t = \frac{\sum_{i=1}^{N} w_i^{(t)} \delta(C_t(x_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(t)}}$$

- Then $f_t(x_i) = f_{t-1}(x_i) + \beta_t C_t(x_i)$

$$w_i^{(t+1)} = \exp[-y_i f_t(x_i)] = w_i^{(t)} e^{-\beta_t y_i C_t(x_i)} = w_i^{(t)} e^{\alpha_t \delta(C_t(x_i) \neq y_i)} e^{-\beta_t}$$

where $\alpha_t = 2\beta_t$ defined at step (d) in discrete AdaBoost algorithm

- This implies discrete AdaBoost algorithm is equivalent to the forward-stagewise additive modeling with exponential loss function.

Why Exponential Loss?

- Logistic regression assumes: $y \in \{1, -1\}$

$$P(y = 1|x) = \frac{1}{1 + \exp(-f(\boldsymbol{x}))} \text{ where } f(\boldsymbol{x}) = \boldsymbol{x}'\boldsymbol{b}$$

- And try to maximize the likelihood: $\text{like} = \prod_{i=1}^{N} \frac{1}{1+\exp(-y_i f(x_i))}$

- Same as to minimize the loss: $\text{Loss} = \sum_{i=1}^{N} \ln(1 + \exp(-y_i f(x_i)))$

- AdaBoost minimizes similar loss: $\text{Loss} = \sum_{i=1}^{N} \exp(-y_i f(x_i))$
  - But $f_T(x) = \sum_{t=1}^{T} \beta_t b(x; \gamma_t)$, where weights β_t is learnt incrementally

Gradient Boost

- Gradient Boost:
  - YuTube: https://www.youtube.com/watch?v=3CC4N4z3GJc
- Algorithm:

1. Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma)$

2. For $t = 1$ to T

   (a) Compute $g_{it} = \left.\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right|_{f(x_i)=f_{t-1}(x_i)}, i = 1, \ldots, N$

   (b) Fit a regression tree to the response g_{it} giving terminal regions $R_{jt}, j = 1, \ldots, J_t$

   (c) Compute $\gamma_{jt} = \underset{\gamma}{\operatorname{argmin}} \sum_{i \in R_{jt}} L(y_i, f_{t-1}(x_i) + \gamma), j = 1, \ldots, J_t$

   (d) Update $f_t(x) = f_{t-1}(x) + \sum_{j=1}^{J_t} \gamma_{jt} \delta(x \in R_{jt})$

3. Output $f_T(x)$ and the classifier is $\operatorname{sign}[f_T(x)]$

How do Bagging and Boosting affect Bias and Variance?

- No general theory
- Some recent experimental studies conclude:
  - Bagging is assumed to reduce variance without changing the bias. However, it has been found to reduce the bias as well, for example, for high-bias classifiers.
  - Boosting has different effects at its different stages. At its early iterations, boosting primarily reduces bias while at the later iterations, it has been found to reduce mainly variance.

Which is better: Bagging or Boosting?

- No “best” method for all problems
- Computational considerations
- Interpretable models
- Some large-scale experiments revealed that
  - Boosting has lower testing error
  - Although Boosting algorithms have been crowned as the “most accurate available off-the-shelf classifiers on a wide variety of data sets”, they are sensitive to noise and outliers, especially for small data sets.
  - Variants of bagging, for example, random forests, have been found to be comparable to AdaBoost.

References

- Breiman, L. (1996) Bagging predictors. *Machine Learning*, 24(2), 123-140.
- Freund, Y. & Schapire, R.E. (1996) Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, Morgan Kauffman, San Francisco, 148-156.
- Friedman, J.H., Hastie, T. and Tibshirani, R. (2000) Additive logistic regression: a statistical view of boosting (with discussion), *Annals of Statistics*, 28, 337-307.
- Friedman, J.H. (2001) Greedy function approximation: the gradient boosting machine, *Annals of Statistics*, 29, 1189-1232.