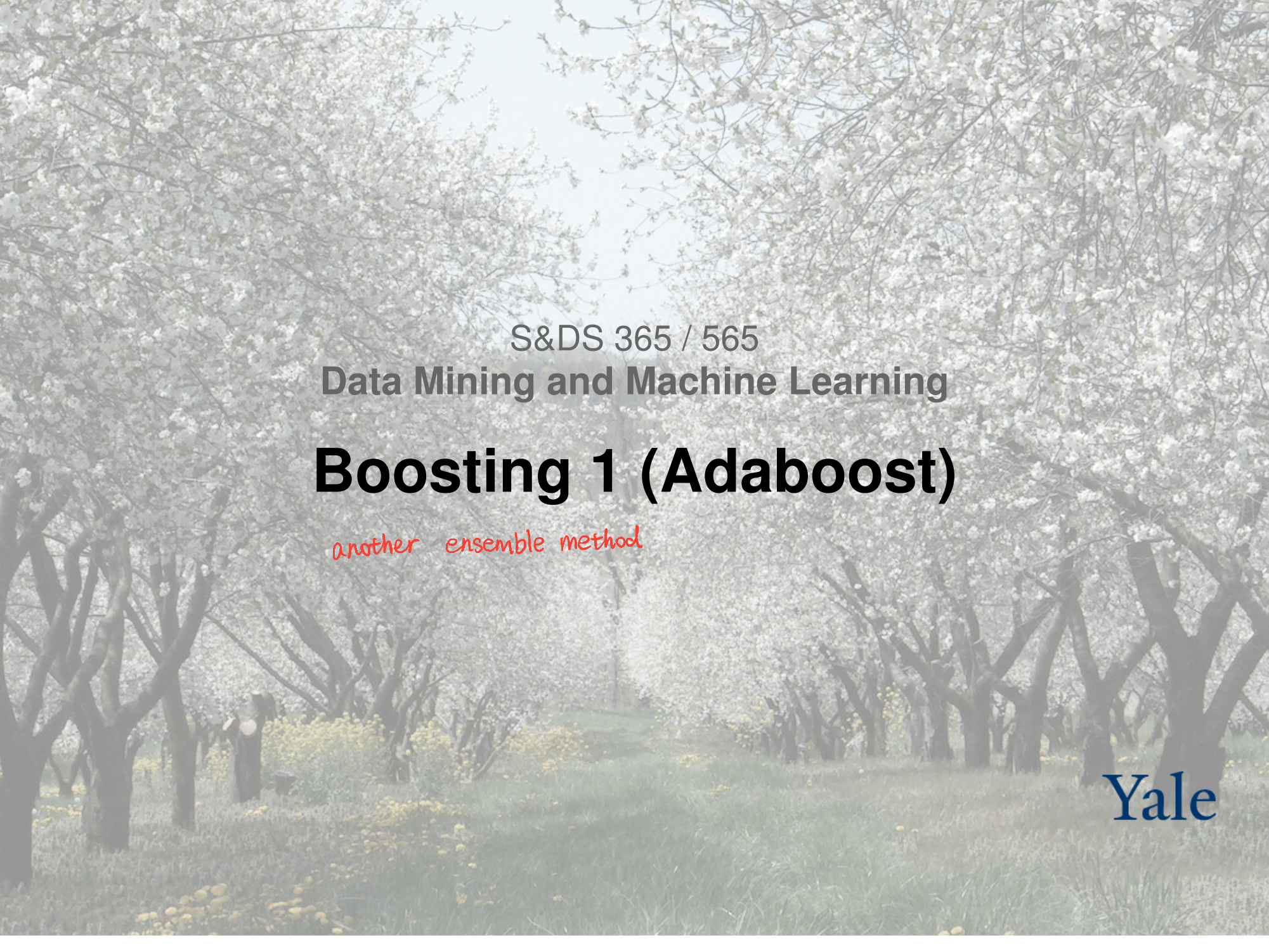

S&DS 365 / 565
Data Mining and Machine Learning
Boosting 1 (Adaboost)
another ensemble method
Yale

Ensembles: Parallel vs Sequential

- Ensemble methods combine multiple models
- Parallel ensembles: each model is built independently
  - e.g. bagging and random forests
  - Main Idea: Combine many (high complexity, low bias) models to reduce variance
- Sequential ensembles: Models are generated sequentially dependently previous model performance based on
  - Try to add new models that do well where previous models lack
    - Weak learner: low complexity

Ensembles: Parallel vs Sequential

"Wisdom of the Crowd" vs "Learn from your mistakes"

parallel

sequential

more like optimization approach

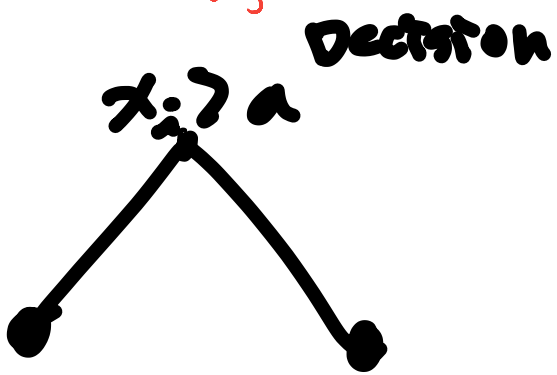
see in boosting part 2

The Boosting Question: Weak learners

use weak learner but still have a better performance

- A weak learner is a classifier that does slightly better than random.
- Weak learners are like “rules of thumb”:
  - If an email has “Viagra” in it, more likely than not it’s spam.
  - Email from a friend is probably not spam.

def decision stump 决策树桩 : a tree with 1 decision rule



Often we use a tree of depth 5
but a decision stump (a tree of depth 1)
is a good place to start

The Boosting Question: Weak learners

激发，促进增长

- A weak learner is a classifier that does slightly better than random.
- Weak learners are like “rules of thumb”:
  - If an email has “Viagra” in it, more likely than not it’s spam.
  - Email from a friend is probably not spam.
- Can we combine a set of weak classifiers to form a single classifier that makes accurate predictions? (Can we extract wisdom from a committee of fools? i.e. how to interpret your current political experiment, maybe)
- Yes! Boosting solves this problem (Rob Schapire (1990))

Restart your mind

For this section, we will think of binary classification in terms of $Y \in \{-1, 1\}$!

A little bit like how going from R to Python and back is (really) annoying, because you have to switch between 1-index to 0-index for arrays.

Adaboost

- Basic idea: Many "weak classifiers" are combined. Data is iteratively reweighted, assigning more weight to examples that have errors. A new component is estimated on the weighted data.
- Each weak classifier $G_m(x)$ outputs a class label $\{-1, 1\}$.
  - Typically, use decision stumps as the weak classifier (tree with a single split).
- Produces a sequence of weak classifiers $G_m(x)$, $m = 1, 2, \ldots$ with weights α_m. Final classifier is

a linear additive model
combine many weak classifiers in a linear way ① like linear regression model

$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

↑ weight

② like random forest
$\alpha_m = \frac{1}{M}$
$G_m(x)$: very complex tree

Boosting idea

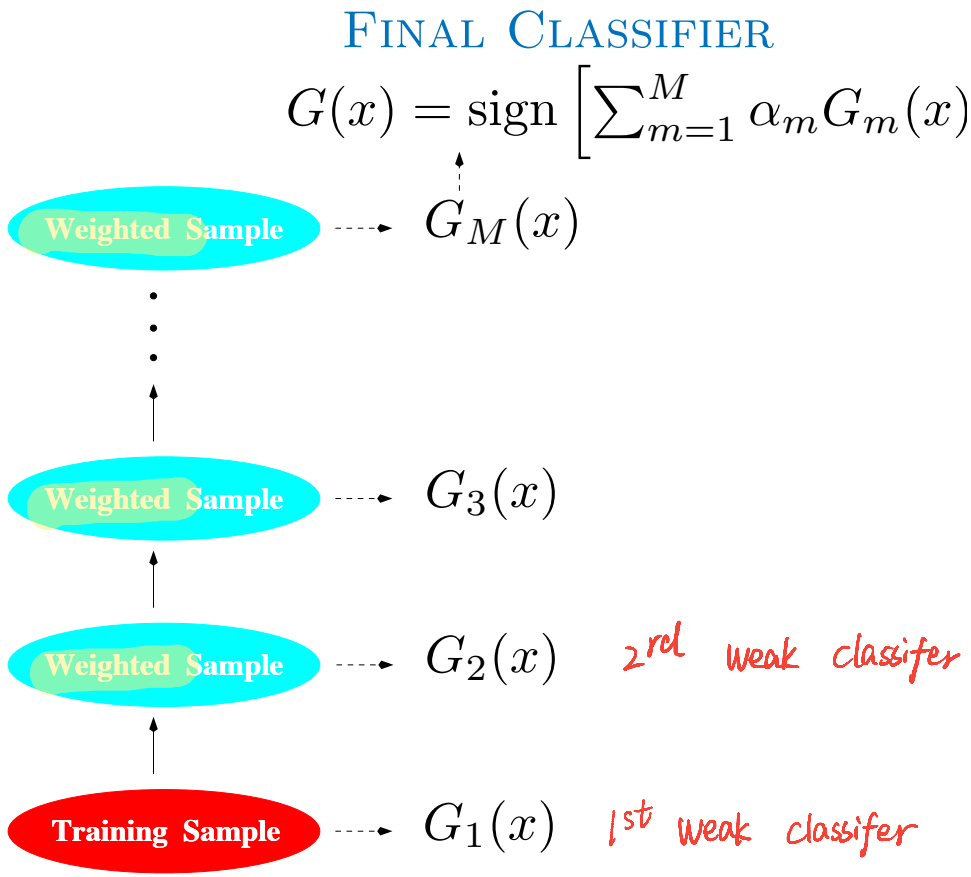


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Boosting idea

Weighting Steps

- Weights $w_1, w_2, \ldots, w_n$ are assigned to examples (x_i, y_i) in each iteration.

totally n examples assign n weights

- Start with $w_i = 1/n$.

equal weights

- At step m, observations that were misclassified by $G_{m-1}(x)$ at the previous step have weights increased.

by exp(α_m)

- Each successive classifier is forced to concentrate on observations that were missed by previous classifiers.

How to use the weights?

even weights $\frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i))$

e.g. loss

$L(y, \hat{y}) = \mathbb{1}(y \neq \hat{y})$

- Weighted empirical risk:
  $\frac{1}{W}\sum_{i=1}^{n} w_i L(y_i, f(x_i))$ where $W = \sum_{i=1}^{n} w_i$
  (normalize, weight, loss)
- Can train a model to minimize weighted empirical risk.
- What if model cannot conveniently be trained to reweighted data?
- Can sample a new data set from training set with probabilities $(w_1/W, \ldots w_n/W)$.

AdaBoost for classification

Given training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ with n examples

1. Initialize weights to $w_i = 1/n$ even weights
2. For $m = 1$ to M:
   - **(a)** Fit classifier $G_m(x)$ to training data using weights w_i.
   - **(b)** Compute weighted error (empirical risk)

$$G_m = \underset{G \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{W} \sum_{i=1}^{n} w_i L(y_i, G(x_i))$$

$\mathcal{F}$: function class; decision stump

minimize

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i \mathbb{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i}$$

   - **(c)** Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
   - **(d)** Set update w_i for next iteration

$$w_i = w_i \cdot \exp\left(\alpha_m \cdot \mathbb{1}(y_i \neq G_m(x_i))\right)$$

3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

← This structure makes cross validation easy for M, coz you can set M to be very large

choice of M: ① large M ② $M = \sqrt{n}$

because for same M, $G_m(x)$ are same, you don't need retrain trees ← just use the trees up to some points that you want to check to validate (eg. k)

AdaBoost: Classifier Weights

- The weight of classifier $G_m(x)$ is $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$.
- Note that weight $\alpha_m \to 0$ as weighted error $\text{err}_m \to 0.5$ (random guessing). decision stump can no longer effectively learn from data

(classifier weight)

Weighted error

AdaBoost: Example Reweighting

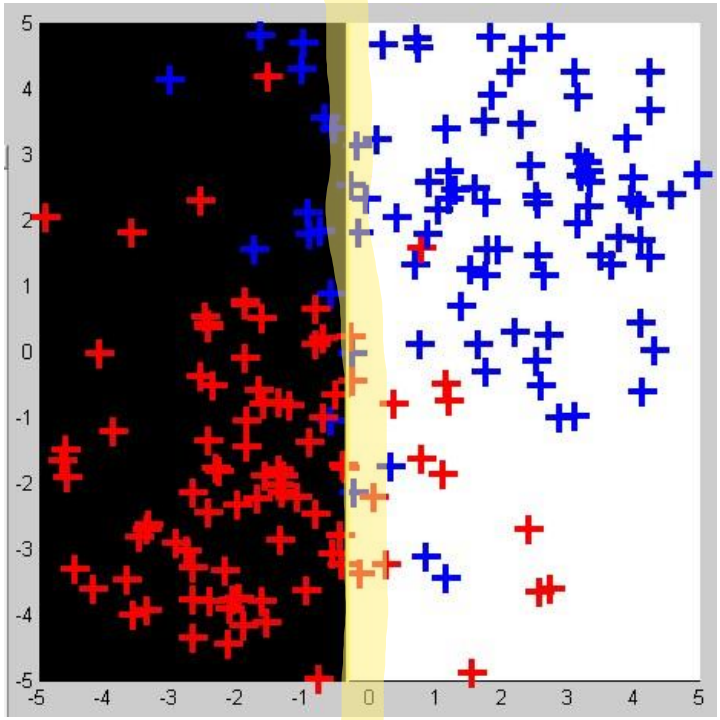
- We train G_m to minimize weighted error, and it achieves err_m.
- Then $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ is the weight of G_m in final ensemble.
- Suppose w_i is weight of example i before training:
  - If G_m classfies x_i correctly, then w_i is unchanged.
  - Otherwise, w_i is increased as $w_i \leftarrow w_i e^{\alpha_m} = w_i \left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
- For $\text{err}_m < 0.5$, this always increases the weight.

AdaBoost with Decision Stump

2 class : red / blue

- After 1 round:

← 1st decision boundary

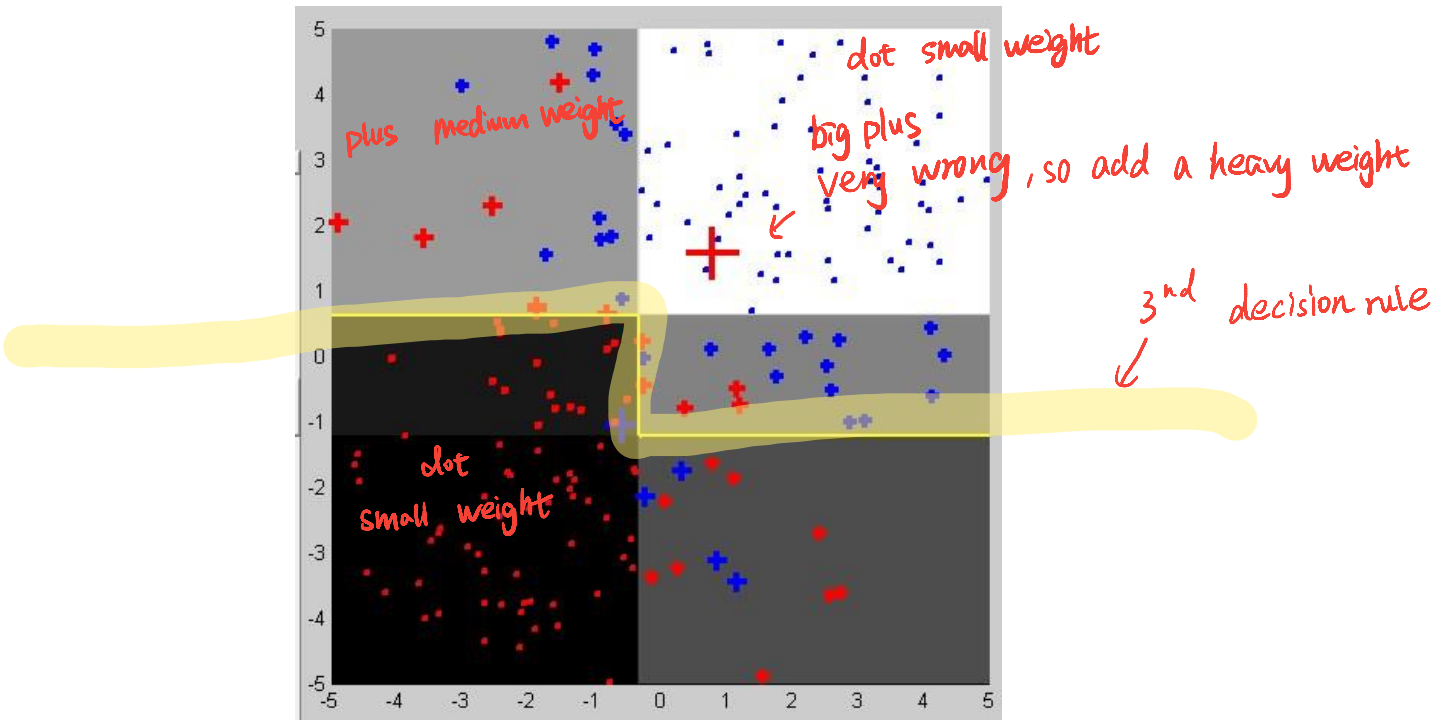


yellow line : decision boundary

Plus size represents weight. Blackness represents score for red class.

AdaBoost with Decision Stumps

- After 3 rounds:

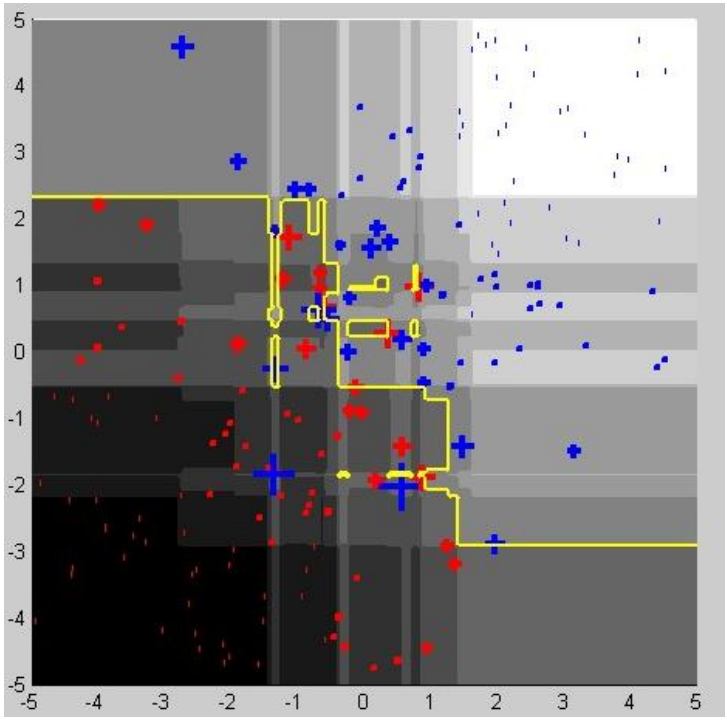


Plus size represents weight. Blackness represents score for red class.

AdaBoost with Decision Stumps

- After 120 rounds:

decision boundary can ← be jagged



Plus size represents weight. Blackness represents score for red class.

Boosting example

Output is

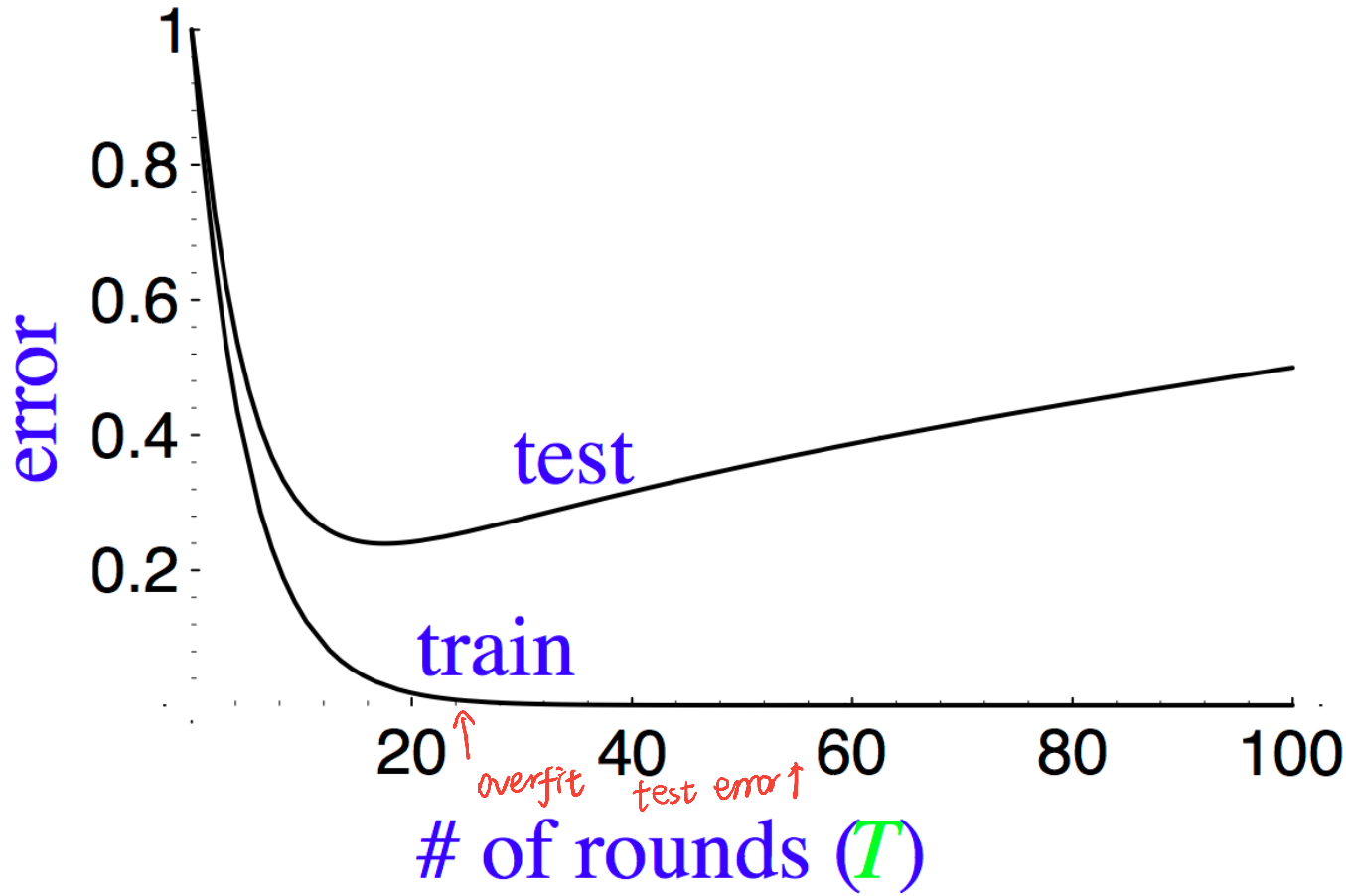
$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j^2 > 9.34 \\ -1 & \text{otherwise} \end{cases}$$

with $X_j \sim N(0,1)$, for $j = 1, \ldots, 10$.

The value 9.34 is the median of the sum of squares of 10 standard Gaussians.

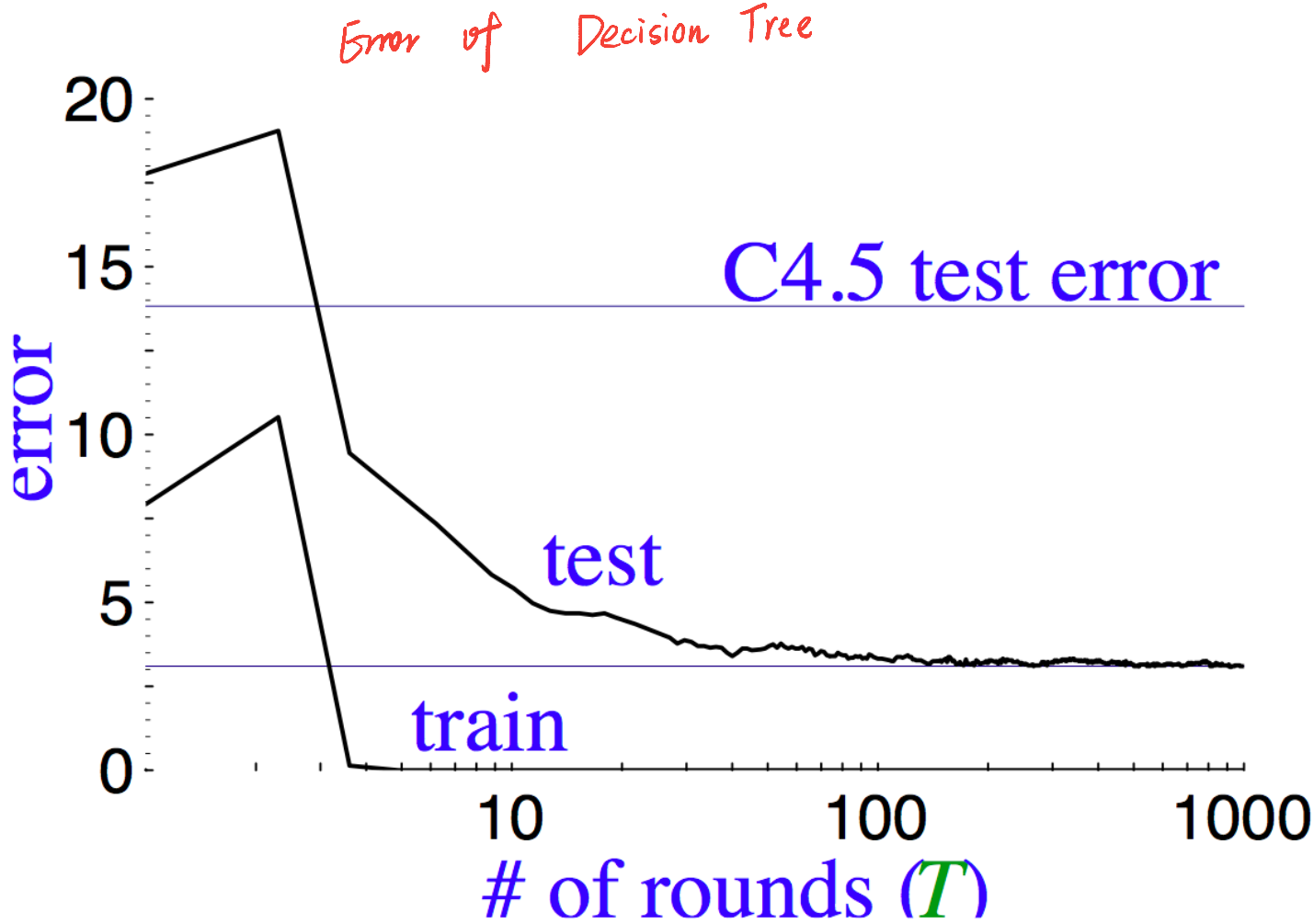
Typical Train / Test Learning Curves

- Might expect too many rounds of boosting to overfit:



Learning Curves for AdaBoost

- In typical performance, AdaBoost is surprisingly resistant to overfitting.
- Test continues to improve even after training error is zero!



Boosting example

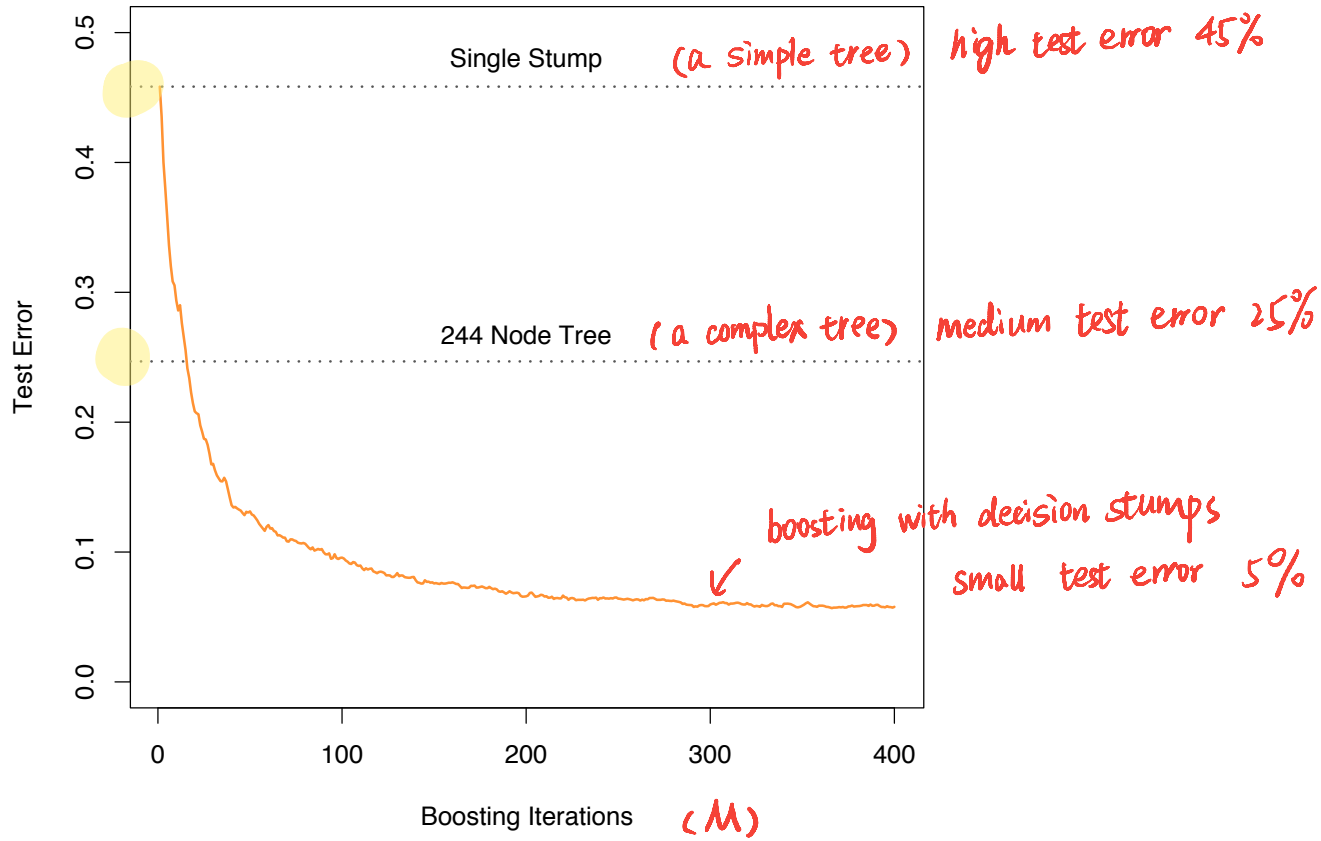


FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Additive Models

- Recall that AdaBoost produces a "score" function of the following form:

$$\sum_{m=1}^{M} \alpha_m G_m(x)$$

More general

same

eg. learners are decision stumps
γ are location value of a feature of splits

- Suppose we have a set of candidate learners $b(x;\gamma)$, indexed by γ. Then, we could similarly consider all functions of this form:

$$f_{\alpha,\gamma}(x) := \sum_{m=1}^{M} \alpha_m b(x;\gamma_m)$$

- Then, simply solve:

① $$\arg\min_{\alpha,\gamma} \sum_{i=1}^{n} L(y_i, f_{\alpha,\gamma}(x_i))$$

- Unfortunately, computationally prohibitive. What to do?

solving ① is complex

How Adaboost address this?

Forward stagewise additive modeling

Adaboost address previous slide problem by ↗

Let's be greedy instead!

rather than directly learn α and γ, learn β and γ (every iteration, try to correct $f_{m-1}(x)$)

- Add a component function $b(x;\gamma)$ with weight β
- $f_m(x) = f_{m-1}(x) + \beta b(x;\gamma)$
- The parameters γ and weight β are optimized in a greedy fashion:

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L\Big(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)\Big)$$

- The loss function $L(y, f)$ is chosen according to the problem

Loss is not misclassification error

Forward stagewise additive modeling

- For example, if $L(y, f(x)) = (y - f(x))^2$ is the squared error, then

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (\overbrace{y_i - f_{m-1}(x_i)}^{r_i} - \beta b(x_i, \gamma))^2 \\ &= (r_i - \beta b(x_i; \gamma))^2 \end{aligned}$$

then we could ignore y and f, just try to use β, γ to fit the r

  where r_i is the residual on the ith data point.

- This is a greedy procedure. We never go back and update the previous functions. 只加 $b(x_i;\gamma)$ 不会减 $b(x_1;\gamma)$
- However, the same function could be added in again

if $f_{m-1}(x_i)$ is not too bad, $f_m(x)$ could be $f_m(x) = f_{m-1}(x_i) + \beta f_{m-1}(x_i)$

Forward stagewise with exponential loss

Boosting uses the exponential loss function

$y = \{-1, +1\}$

$f(x)$ very positive $\begin{cases} y=+1 & L \text{ very small} \\ y=-1 & L \text{ very large} \end{cases}$

$$L(y, f(x)) = \exp(-yf(x))$$

The quantity $yf(x)$ is sometimes called *the margin* (How far away point is correctly labeled)

The additive modeling strategy is:

optimize this yields same result as previous Adaboost using misclassification error

$$\begin{aligned} (\beta_m, G_m) &= \arg\min_{\beta, G} \sum_{i=1}^{n} \exp\left[-y_i \left(f_{m-1}(x_i) + \beta G(x_i)\right)\right] \\ &= \arg\min_{\beta, G} \sum_{i=1}^{n} \overbrace{w_i^{(m)}}^{\text{weight}} \exp\left(-\beta y_i G(x_i)\right) \end{aligned}$$

(β_m: weight; G_m: decision stump function; $w_i^{(m)}$: weighted loss)

where $w_i^{(m)}$ is defined as $\exp(-y_i f_{m-1}(x_i))$.

Forward stagewise with exponential loss

The additive modeling strategy is:

$$(\beta_m, G_m) \quad = \quad \arg\min_{\beta, G} \sum_{i=1}^{n} w_i^{(m)} \exp\left(-\beta y_i G(x_i)\right) \quad ①$$

where $w_i^{(m)}$ is defined as $\exp(-y_i f_{m-1}(x_i))$.

positive

For any value of β, this is the same as minimizing

$y_i = \pm 1$ $G(x_i) = \pm 1$ $\Rightarrow \exp(-\beta y_i G(x_i)) = \begin{cases} \exp(-\beta) & y_i = G(x_i) \\ \exp(\beta) & y_i \neq G(x_i) \end{cases}$ $\Rightarrow$ factor $\exp(\pm\beta)$ out

$$G_m = \arg\min_{G} \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G(x_i)) \quad ②$$

weight 0-1 loss

Mean find G_m is independent of β

just minimizing a weighted loss

So we can find G_m first use ②, plug G_m back in ①, then find β_m

Forward stagewise with exponential loss

① $\sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i G(x_i))$

To see this, write the expression as

$$e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

$$= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}$$

if $\beta > 0$, this term > 0

Thus, minimize ① = minimizing 黄框

this term independent of G

↓

G_m

Forward stagewise with exponential loss

Plugging this G_m back in and solving for β_m, some algebra yields

$$\beta_m = \frac{1}{2}\log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$$

where err_m is the weighted error rate

$$\text{err}_m = \frac{\sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{n} w_i^{(m)}}$$

The weights are then updated as

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)},$$

or refactorizing

$$w_i^{(m+1)} = w_i^{(m)} \cdot e^{2\beta_m I\{y_i \neq G_m(x_i)\}} e^{-\beta_m}$$ → for scaling, can ignore

Recall previous AdaBoost $w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m} = w_i^{(m)} \cdot e^{2\beta_m}$

$\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$

same algorithm

$\sum_{i=1}^{n} w_i^{(m)} \exp(-\beta y_i G(x_i))$

now just scalar optimization

$\text{Loss} = (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i^{(m)} \mathbb{1}(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{n} w_i^{(m)}$

$\beta_m = \arg\min_{\beta} \text{Loss}$

$\min(\text{Loss}) = 0$

$(e^{\beta} - e^{-\beta}) \frac{\sum w_i^{(m)} \mathbb{1}(y_i \neq G(x_i))}{\sum_{i=1}^{n} w_i^{(m)}} + e^{-\beta} = 0$

$(e^{\beta} - e^{-\beta})\,\text{err}_m + e^{-\beta} = 0$

$(e^{2\beta} - 1)\,\text{err}_m + 1 = 0$

$e^{2\beta} = \frac{-1}{\text{err}_m} + 1$

$2\beta = \ln\left(1 - \frac{1}{\text{err}_m}\right)$

$\beta = \frac{1}{2}\ln\frac{\text{err}_m - 1}{\text{err}_m}$

Simulation

proof exponential loss is a good surrogate for misclassification loss (0-1 loss)

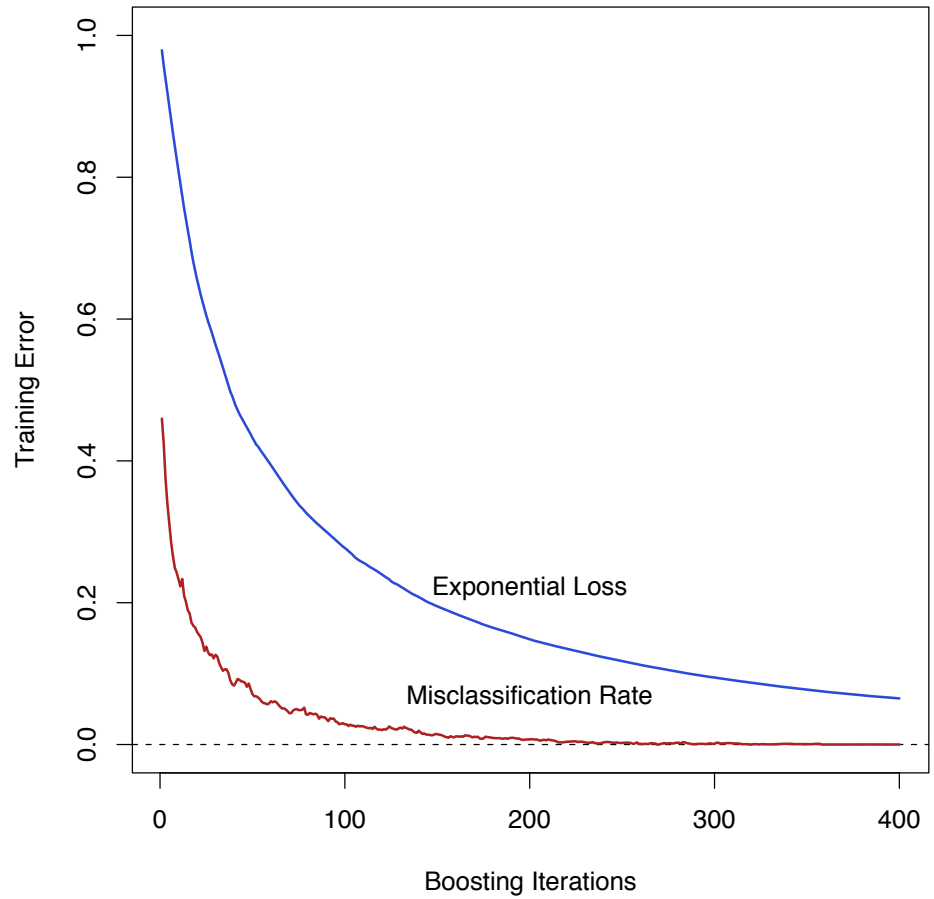


FIGURE 10.3. *Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss:* $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$. *After about 250 iterations, the misclassification error is zero, while the exponential loss continues to decrease.*

Performance on email spam task

(classification and regression tree) / (decision tree)

CART: 8.7%

Additive logistic regression: 5.5%

Random forests: 5.1%

Boosting: 4.5% Best !

(standard error of the estimates is $\approx 0.6\%$)

Loss functions

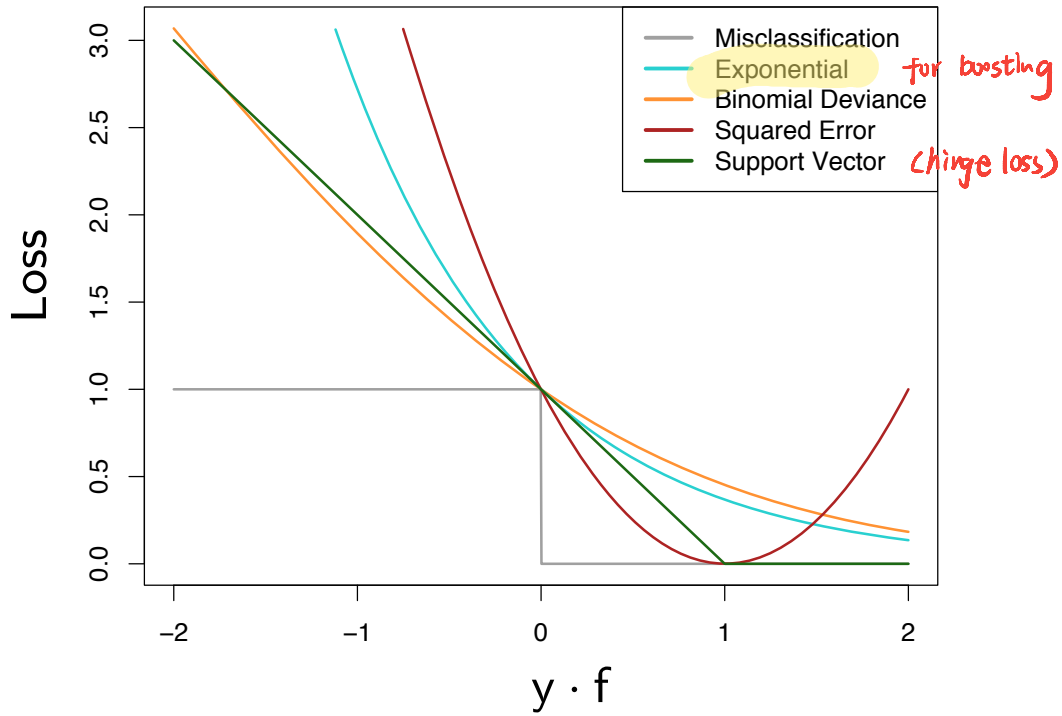


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*