Boosting

Trevor Hastie
Statistics Department
Stanford University

Collaborators: Brad Efron, Jerome Friedman, Saharon Rosset, Rob Tibshirani, Ji Zhu

`http://www-stat.stanford.edu/∼hastie/TALKS/boost.pdf`

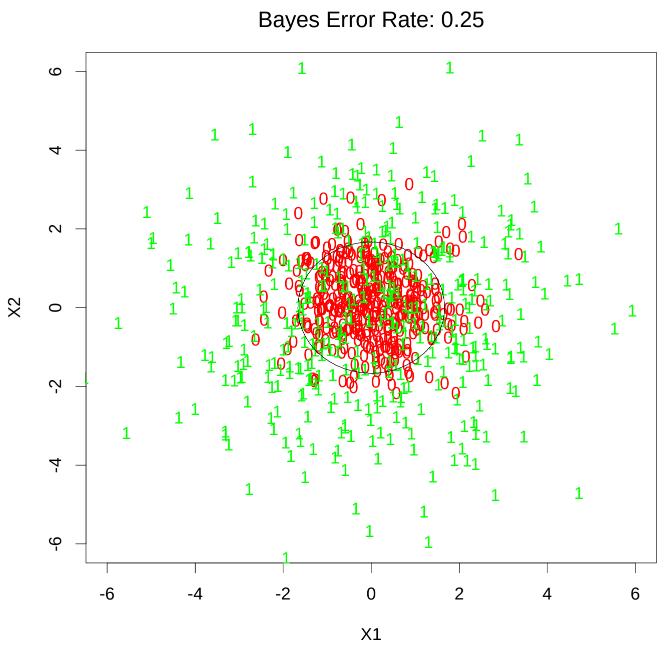
Outline

- Model Averaging
  - Bagging
  - Boosting
- History of Boosting
- Stagewise Additive Modeling
- Boosting and Logistic Regression
- MART
- Boosting and Overfitting
- Summary of Boosting, and its place in the toolbox.

Methods for improving the performance of weak learners. Classification trees are adaptive and robust, but do not generalize well. The techniques discussed here enhance their performance considerably.

Reference: Chapter 10.

Classification Problem



Data $(X, Y) \in R^p \times \{0, 1\}$.

X is predictor, feature; Y is class label, response.

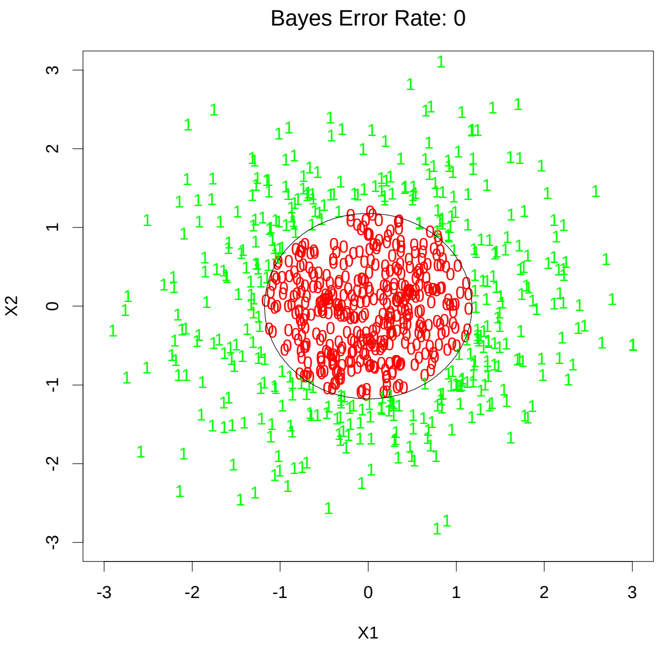
(X, Y) have joint probability distribution $\mathcal{D}$.

Goal: Based on N training pairs $(X_i,\ Y_i)$ drawn from $\mathcal{D}$ produce a classifier $\hat{C}(X) \in \{0, 1\}$

Goal: choose $\hat{C}$ to have low generalization error

$$\begin{aligned} R(\hat{C}) &= P_{\mathcal{D}}(\hat{C}(X) \neq Y) \\ &= E_{\mathcal{D}}[1_{(\hat{C}(X) \neq Y)}] \end{aligned}$$

Deterministic Concepts



$X \in R^p$ has distribution $\mathcal{D}$.

$C(X)$ is deterministic function $\in$ concept class.

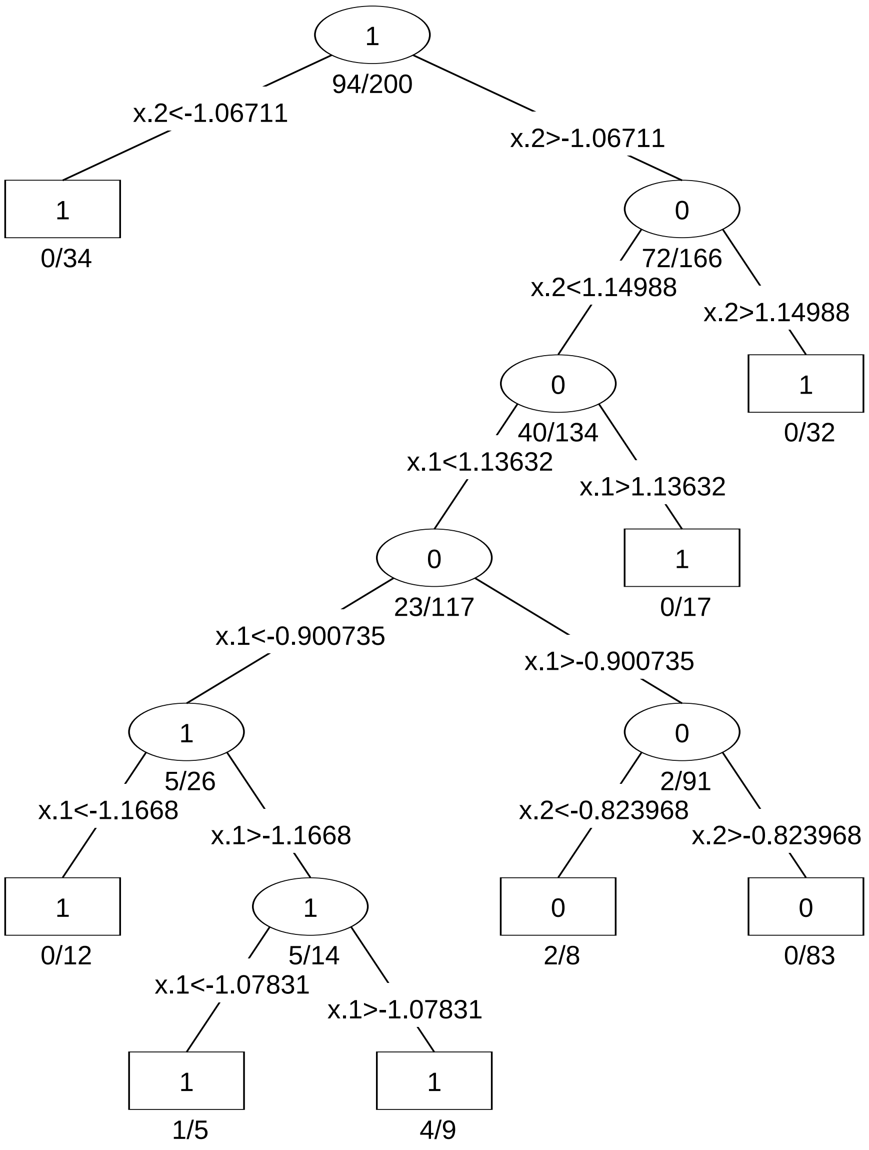
Goal: Based on N training pairs $(X_i,\ Y_i = C(X_i))$ drawn from $\mathcal{D}$ produce a classifier $\hat{C}(X) \in \{0,1\}$

Goal: choose $\hat{C}$ to have low generalization error

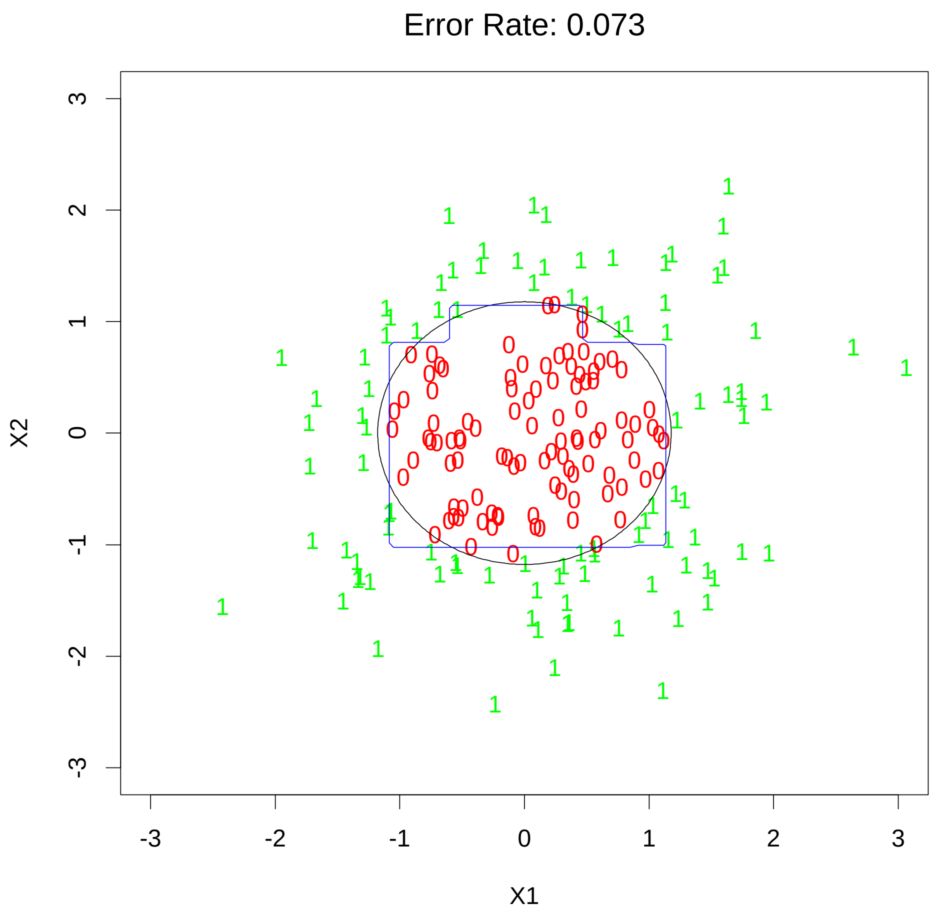
$$\begin{aligned} R(\hat{C}) &= P_{\mathcal{D}}(\hat{C}(X) \neq C(X)) \\ &= E_{\mathcal{D}}[1_{(\hat{C}(X) \neq C(X))}] \end{aligned}$$

Classification Tree

Sample of size 200



Decision Boundary: Tree



When the nested spheres are in $\mathbb{R}^{10}$, CART produces a rather noisy and inaccurate rule $\hat{G}(X)$, with error rates around 30%.

Model Averaging

Classification trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.

In general Boosting $\succ$ Bagging $\succ$ Single Tree.

"AdaBoost $\cdots$ best off-the-shelf classifier in the world" — Leo Breiman, NIPS workshop, 1996.

Bagging

Bagging or bootstrap aggregation averages a given procedure over many samples, to reduce its variance — a poor man's Bayes. See pp 246.

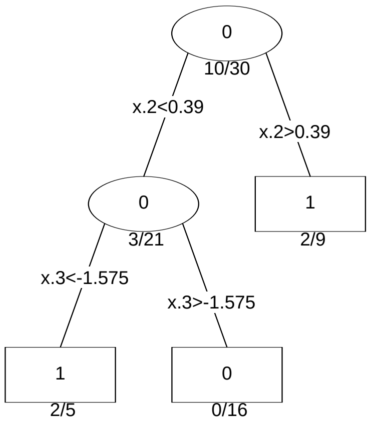
Suppose $G(\mathbf{X}, x)$ is a classifier, such as a tree, producing a predicted class label at input point x.

To bag G, we draw bootstrap samples $\mathbf{X}^{*1}, \ldots \mathbf{X}^{*B}$ each of size N with replacement from the training data. Then

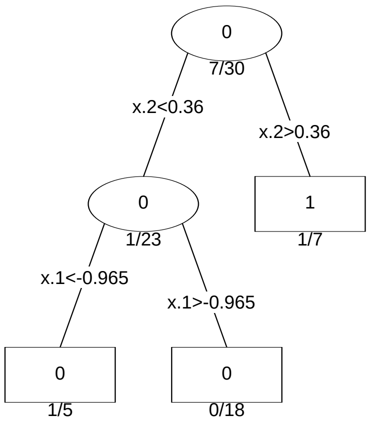
$$\hat{G}_{bag}(x) = \text{Majority Vote } \{G(\mathbf{X}^{*b}, x)\}_{b=1}^{B}.$$

Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction. However any simple structure in G (e.g a tree) is lost.

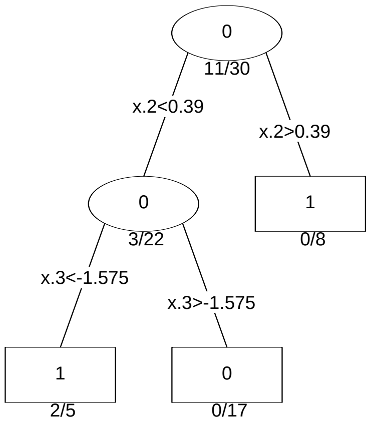
Original Tree



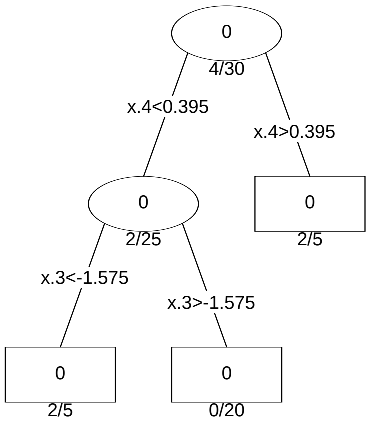
Bootstrap Tree 1



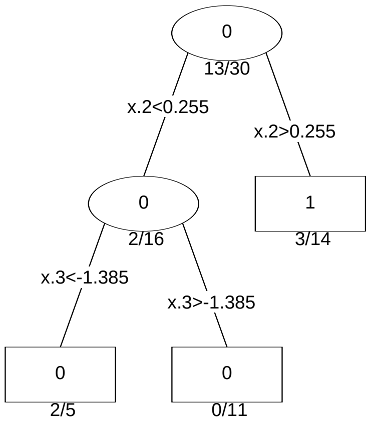
Bootstrap Tree 2



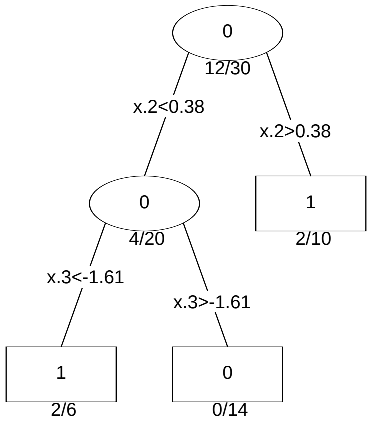
Bootstrap Tree 3



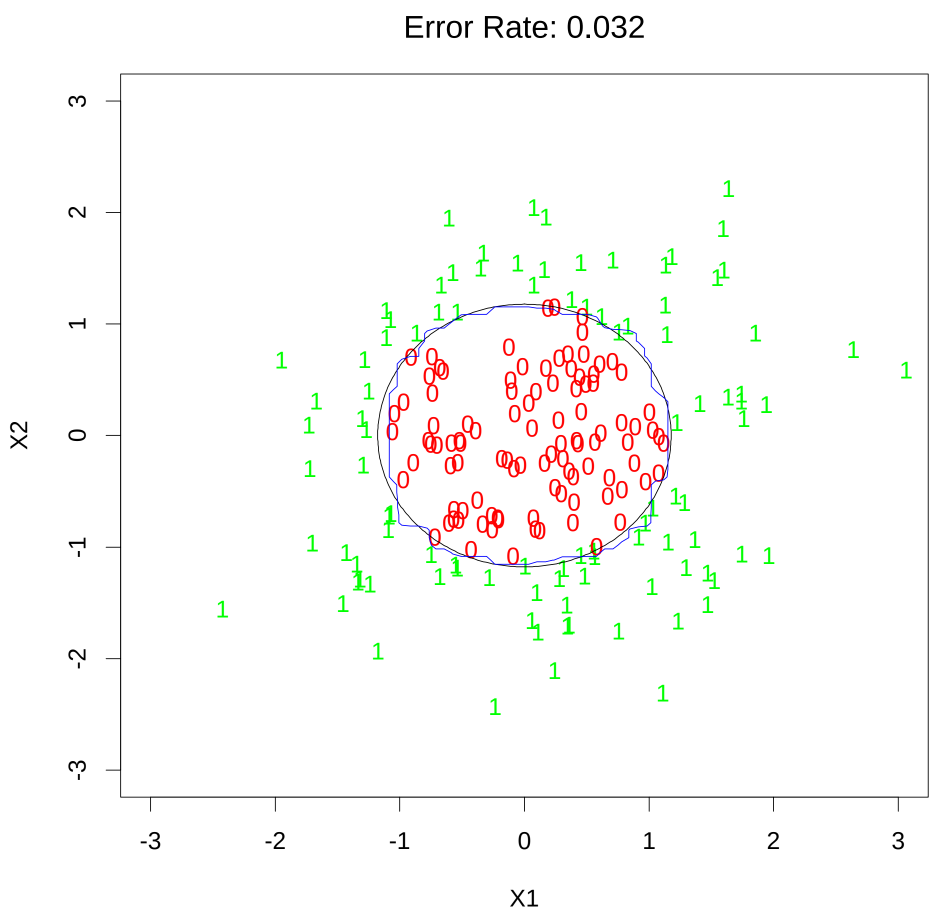
Bootstrap Tree 4



Bootstrap Tree 5

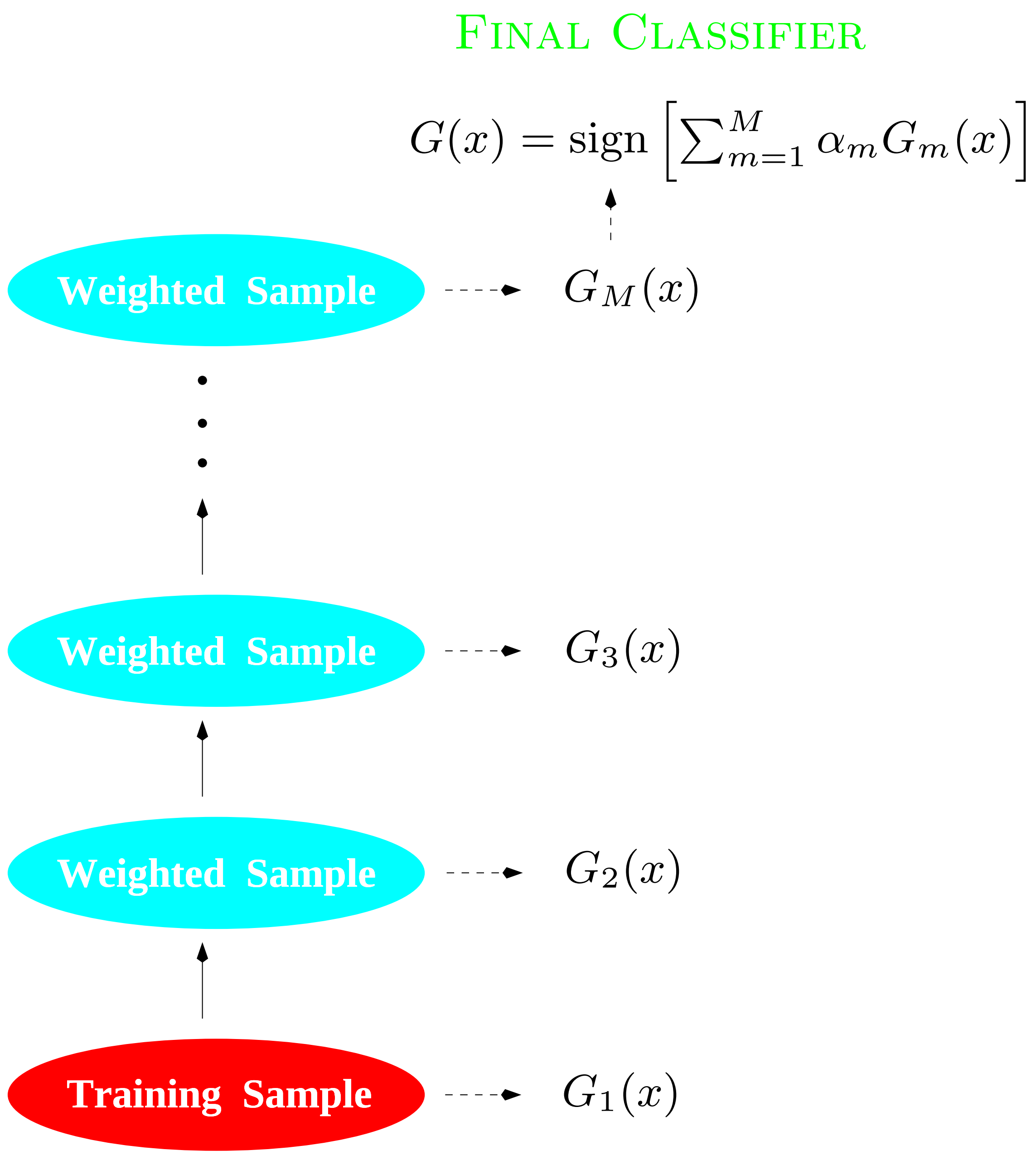


Decision Boundary: Bagging

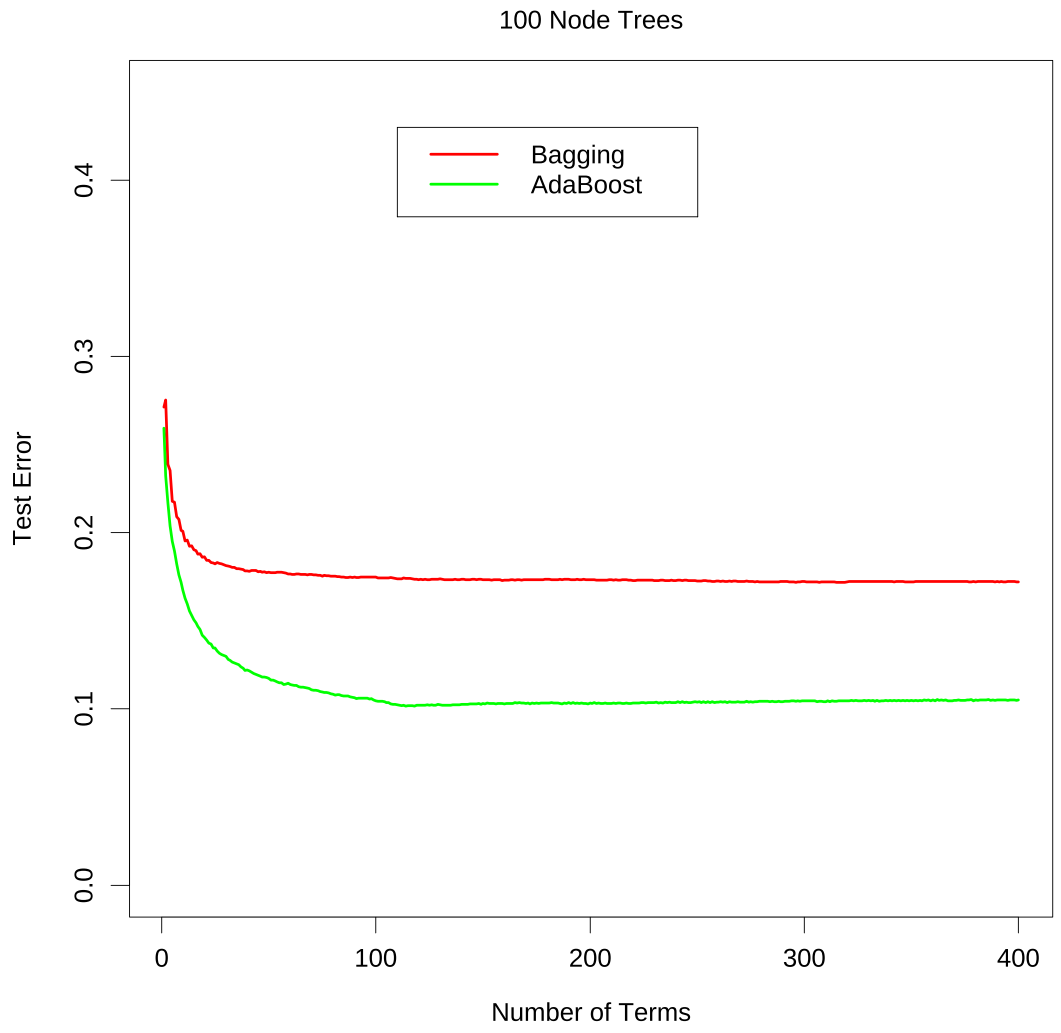


Bagging averages many trees, and produces smoother decision boundaries.

Boosting


FINAL CLASSIFIER
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

Bagging and Boosting



2000 points from Nested Spheres in R^{10}; Bayes error rate is 0%.

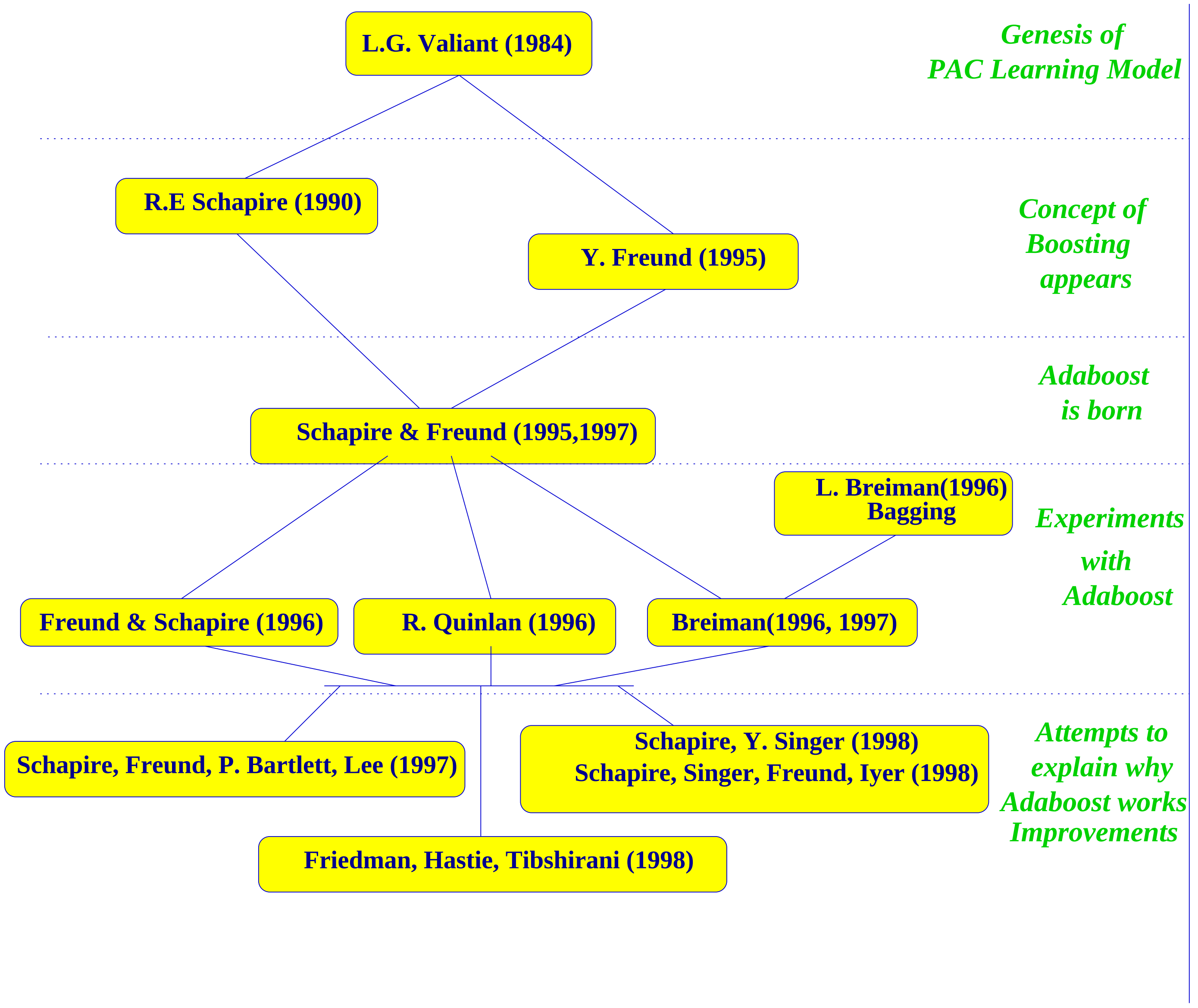
Trees are grown Best First without pruning.

Leftmost iteration is a single tree.

AdaBoost (Freund & Schapire, 1996)

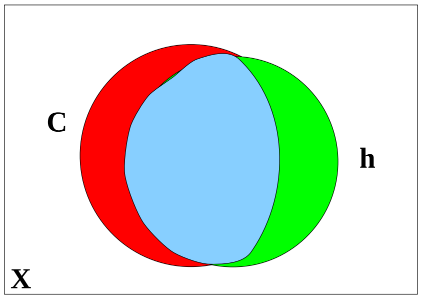
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M repeat steps (a)–(d):
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Update weights for $i = 1, \ldots, N$:
   $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$
   and renormalize to w_i to sum to 1.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.

History of Boosting


L.G. Valiant (1984)
Genesis of PAC Learning Model
R.E Schapire (1990)
Y. Freund (1995)
Concept of Boosting appears
Schapire & Freund (1995,1997)
Adaboost is born
L. Breiman(1996) Bagging
Freund & Schapire (1996)
R. Quinlan (1996)
Breiman(1996, 1997)
Experiments with Adaboost
Schapire, Freund, P. Bartlett, Lee (1997)
Schapire, Y. Singer (1998)
Schapire, Singer, Freund, Iyer (1998)
Friedman, Hastie, Tibshirani (1998)
Attempts to explain why Adaboost works
Improvements

PAC Learning Model

$X \sim \mathcal{D}$: Instance Space
$C : X \mapsto \{0, 1\}$ Concept $\in \mathcal{C}$
$h : X \mapsto \{0, 1\}$ Hypothesis $\in \mathcal{H}$
$\text{error}(h) = P_{\mathcal{D}}[C(X) \neq h(X)]$



Definition: Consider a concept class $\mathcal{C}$ defined over a set X of length N. L is a learner (algorithm) using hypothesis space $\mathcal{H}$. $\mathcal{C}$ is PAC learn-able by $\mathcal{L}$ using $\mathcal{H}$ if for all $C \in \mathcal{C}$, all distributions $\mathcal{D}$ over X and all $\epsilon, \delta \in (0, \frac{1}{2})$, learner L will, with $Pr \geq (1 - \delta)$, output an $h \in \mathcal{H}$ s.t. $\text{error}_D(h) \leq \epsilon$ in time polynomial in $1/\epsilon, 1/\delta, N$ and size($\mathcal{C}$).

Such an L is called a strong Learner.

Boosting a Weak Learner

Weak learner L produces an h with error rate $\beta = (\frac{1}{2} - \varepsilon) < \frac{1}{2}$, with $Pr \geq (1 - \delta)$ for any $\mathcal{D}$.

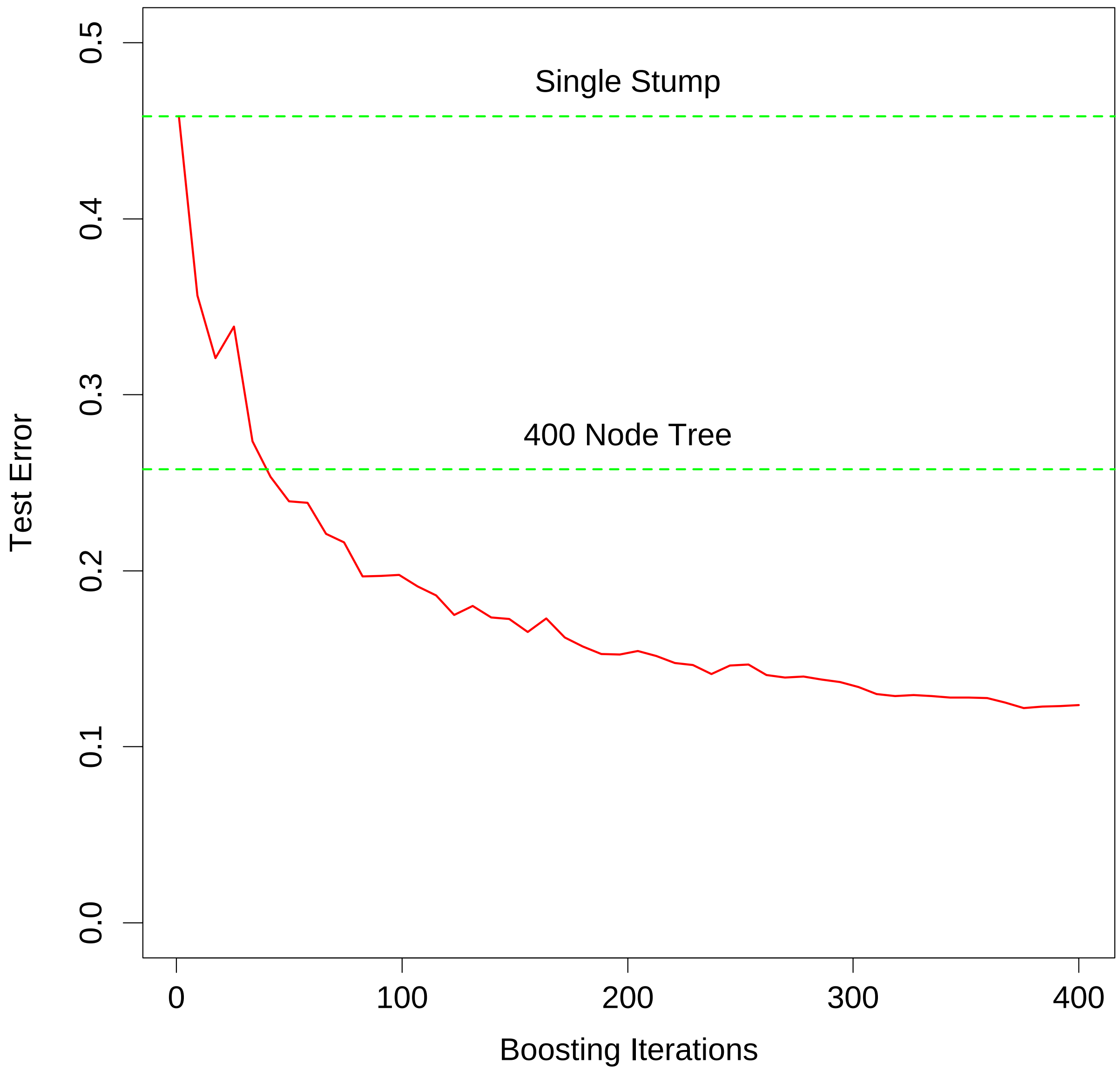
L has access to continuous stream of training data and a class oracle.

1. L learns h_1 on first N training points.
2. L randomly filters the next batch of training points, extracting $N/2$ points correctly classified by h_1, $N/2$ incorrectly classified, and produces h_2.
3. L builds a third training set of N points for which h_1 and h_2 disagree, and produces h_3.
4. L outputs $h = \mathit{Majority\ Vote}(h_1, h_2, h_3)$

THEOREM (Schapire, 1990): "The Strength of Weak Learnability"

$$\text{error}_D(h) \leq 3\beta^2 - 2\beta^3 < \beta$$

Boosting Stumps

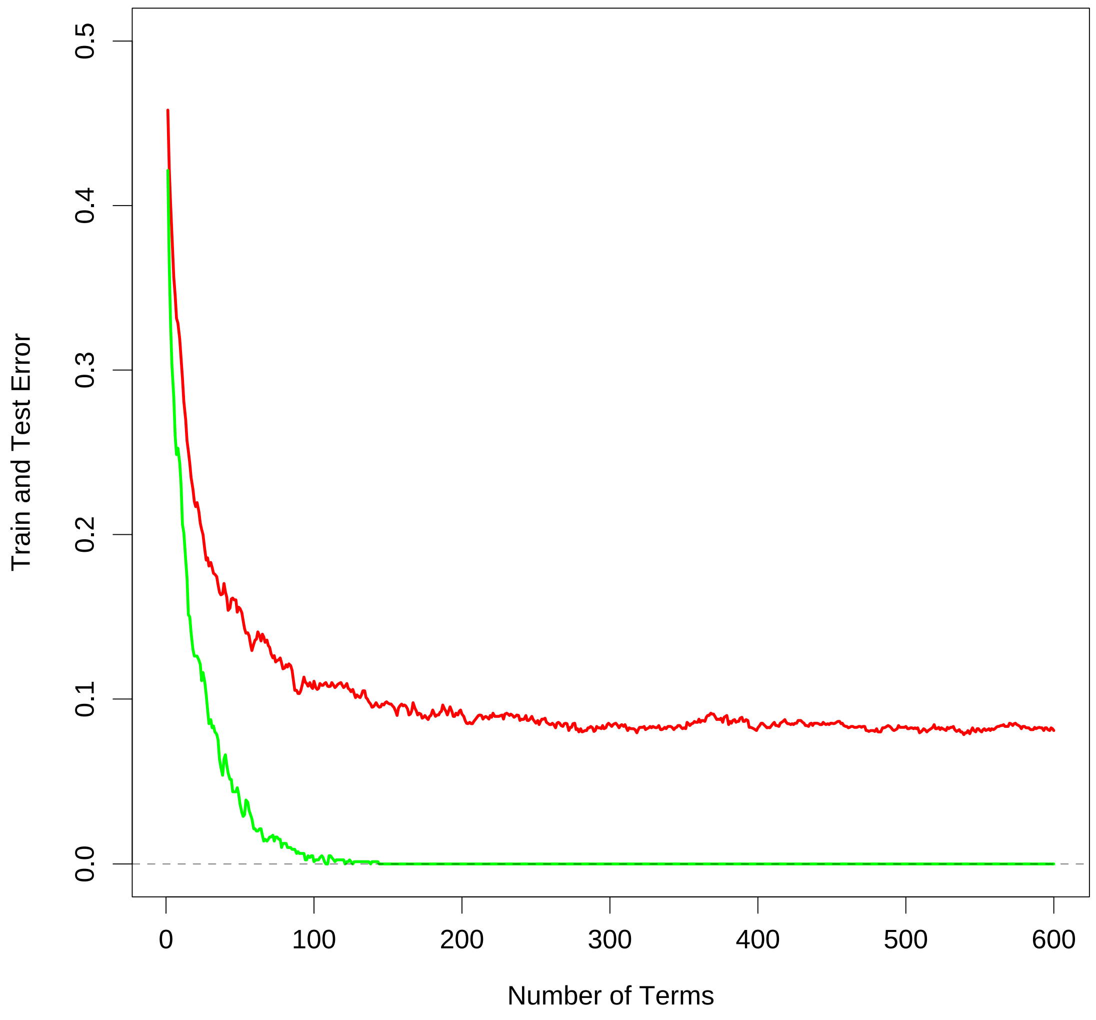


A stump is a two-node tree, after a single split. Boosting stumps works remarkably well on the nested-spheres problem.

Boosting & Training Error

Nested spheres in R^{10} — Bayes error is 0%.

Stumps

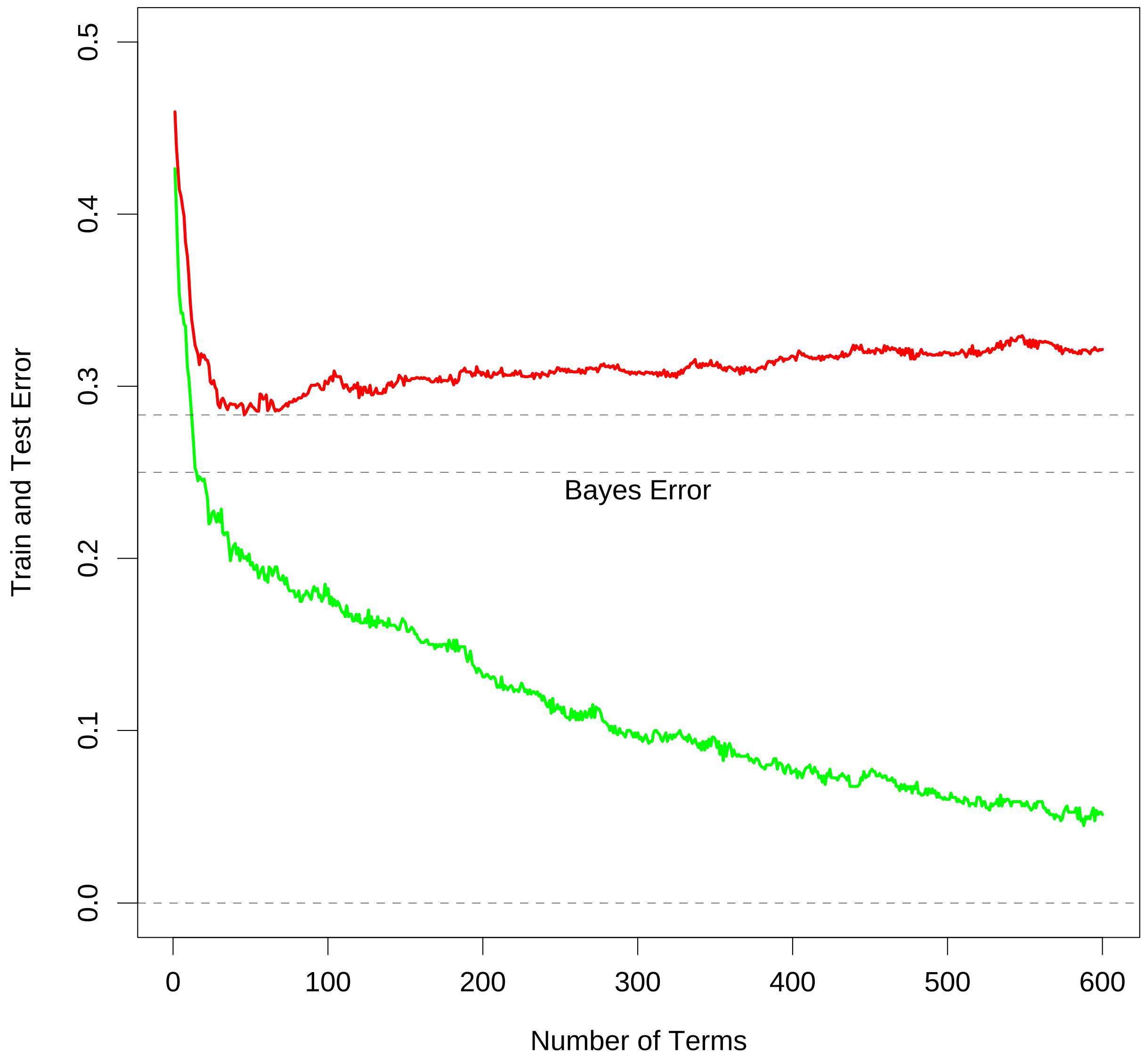


Boosting drives the training error to zero. Further iterations continue to improve test error in many examples.

Boosting Noisy Problems

Nested Gaussians in R^{10} — Bayes error is 25%.

Stumps



Here the test error does increase, but quite slowly.

Stagewise Additive Modeling

Boosting builds an additive model

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m).$$

Here $b(x, \gamma_m)$ is a tree, and γ_m parametrizes the splits.

We do things like that in statistics all the time!

- GAMs: $f(x) = \sum_j f_j(x_j)$
- Basis expansions: $f(x) = \sum_{m=1}^{M} \theta_m h_m(x)$

Traditionally the parameters f_m, θ_m are fit jointly (i.e. least squares, maximum likelihood).

With boosting, the parameters (β_m, γ_m) are fit in a stagewise fashion. This slows the process down, and tends to overfit less quickly.

Stagewise Least Squares

Suppose we have available a basis family $b(x;\gamma)$ parametrized by γ. For example, a simple family is $b(x;\gamma_j) = x_j$.

- After $m-1$ steps, suppose we have the model $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x;\gamma_j)$.
- At the mth step we solve

$$\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2$$

- Denoting the residuals at the mth stage by $r_{im} = y_i - f_{m-1}(x_i)$, the previous step amounts to

$$\min_{\beta,\gamma} (r_{im} - \beta b(x_i;\gamma))^2,$$

- Thus the term $\beta_m b(x;\gamma_m)$ that best fits the current residuals is added to the expansion at each step.

Adaboost: Stagewise Modeling

- AdaBoost builds an additive logistic regression model

$$f(x) = \log \frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)} = \sum_{m=1}^{M} \alpha_m G_m(x)$$

by stagewise fitting using the loss function

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

- Given the current $f_{M-1}(x)$, our solution for (β_m, G_m) is

$$\arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x))]$$

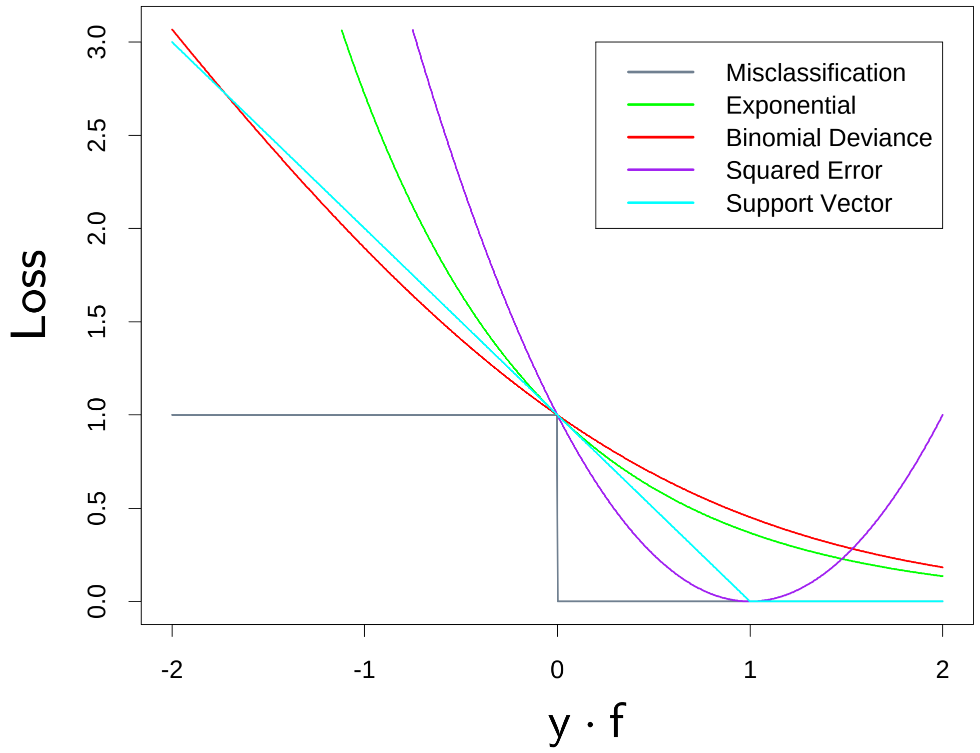
where $G_m(x) \in \{-1, 1\}$ is a tree classifier and β_m is a coefficient.

- With $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$, this can be re-expressed as

$$\arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

- We can show that this leads to the Adaboost algorithm; See pp 305.

Why Exponential Loss?



- $e^{-yF(x)}$ is a monotone, smooth upper bound on misclassification loss at x.
- Leads to simple reweighting scheme.
- Has logit transform as population minimizer

$$f^*(x) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

- Other more robust loss functions, like binomial deviance.

General Stagewise Algorithm

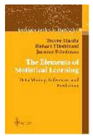
We can do the same for more general loss functions, not only least squares.

```
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   (a) Compute
       $(\beta_m, \gamma_m) =$
       $\arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
```

Sometimes we replace step (b) in item 2 by

```
   (b*) Set $f_m(x) = f_{m-1}(x) + \nu \beta_m b(x; \gamma_m)$
```

Here ν is a shrinkage factor, and often $\nu < 0.1$.

See  pp 326. Shrinkage slows the stagewise model-building even more, and typically leads to better performance.

MART

- General boosting algorithm that works with a variety of different loss functions. Models include regression, outlier-resistant regression, K-class classification and risk modeling.
- MART uses gradient boosting to build additive tree models, for example, for representing the logits in logistic regression.
- Tree size is a parameter that determines the order of interaction (next slide).
- MART inherits all the good features of trees (variable selection, missing data, mixed predictors), and improves on the weak features, such as prediction performance.
- MART is described in detail in , section 10.10.

MART in detail

Model

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

where $T(x; \Theta)$ is a tree:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$

Fitting Criterion

Given $\hat{\Theta}_j,\ j = 1, \ldots, m-1$, we obtain $\hat{\Theta}_m$ via stagewise optimization:

$$\arg\min_{\Theta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)\right)$$

For general loss functions this is a very difficult optimization problem. Gradient boosting is an approximate gradient descent method.

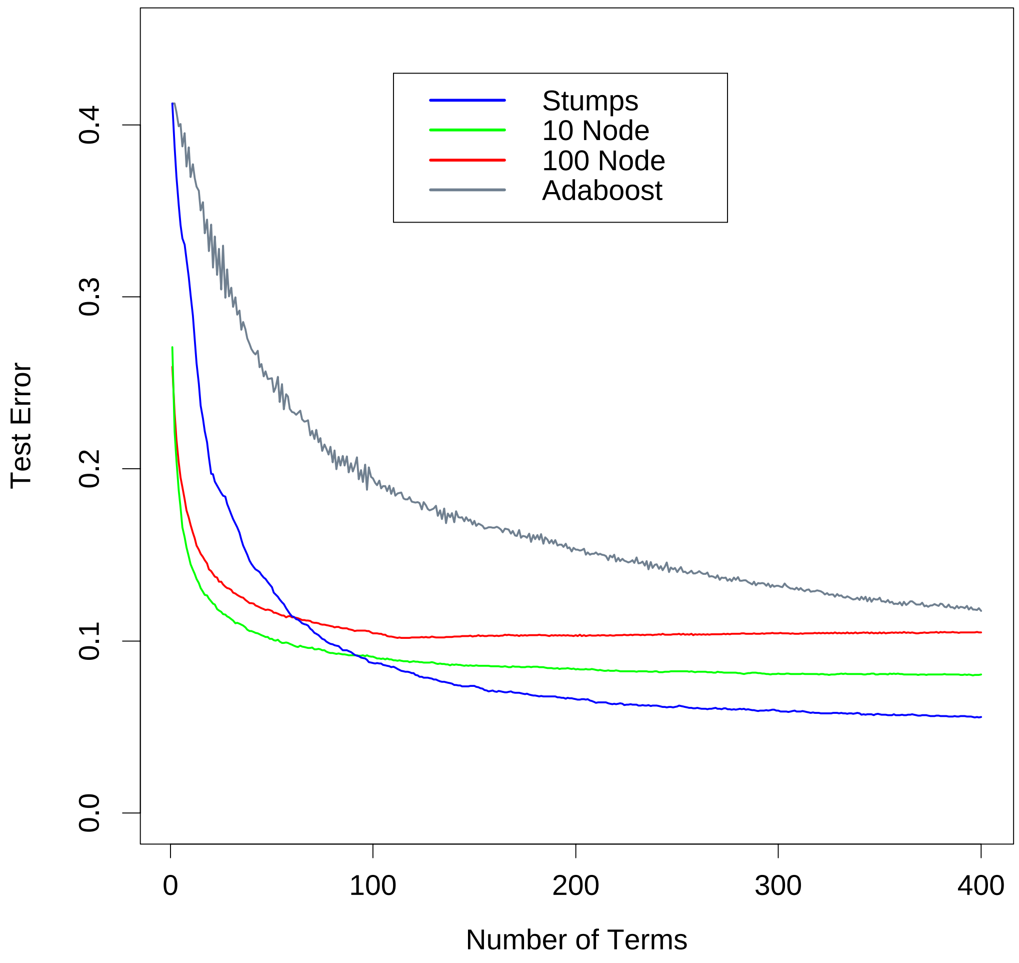
Gradient Boosting

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm},\ j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Tree Size

The tree size J determines the interaction order of the model:

$$\eta(X) \quad = \quad = \sum_j \eta_j(X_j) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) + \cdots$$



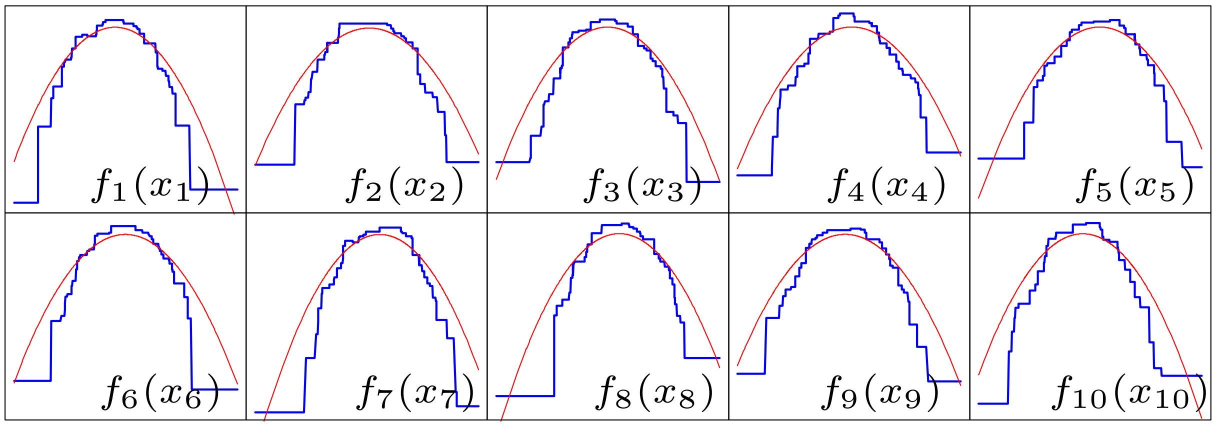
Stumps win!

Since the true decision boundary is the surface of a sphere, the function that describes it has the form

$$f(X) = X_1^2 + X_2^2 + \ldots + X_p^2 - c = 0.$$

Boosted stumps via MART returns reasonable approximations to these quadratic functions.

Coordinate Functions for Additive Logistic Trees



Example: Predicting e-mail spam

- data from 4601 email messages
- goal: predict whether an email message is spam (junk email) or good.
- input features: relative frequencies in a message of 57 of the most commonly occurring words and punctuation marks in all the training the email messages.
- for this problem not all errors are equal; we want to avoid filtering out good email, while letting spam get through is not desirable but less serious in its consequences.
- we coded `spam` as 1 and `email` as 0.
- A system like this would be trained for each user separately (e.g. their word lists would be different)

Predictors

- 48 quantitative predictors—the percentage of words in the email that match a given word. Examples include `business`, `address`, `internet`, `free`, and `george`. The idea was that these could be customized for individual users.
- 6 quantitative predictors—the percentage of characters in the email that match a given character. The characters are `ch;`, `ch(`, `ch[`, `ch!`, `ch$`, and `ch#`.
- The average length of uninterrupted sequences of capital letters: `CAPAVE`.
- The length of the longest uninterrupted sequence of capital letters: `CAPMAX`.
- The sum of the length of uninterrupted sequences of capital letters: `CAPTOT`.

Some important features

39% of the training data were spam.

Average percentage of words or characters in an email message equal to the indicated word or character. We have chosen the words and characters showing the largest difference between spam and email.

	george	you	your	hp	free	hpl
spam	0.00	2.26	1.38	0.02	0.52	0.01
email	1.27	1.27	0.44	0.90	0.07	0.43

	!	our	re	edu	remove
spam	0.51	0.51	0.13	0.01	0.28
email	0.11	0.18	0.42	0.29	0.01

Spam Example Results

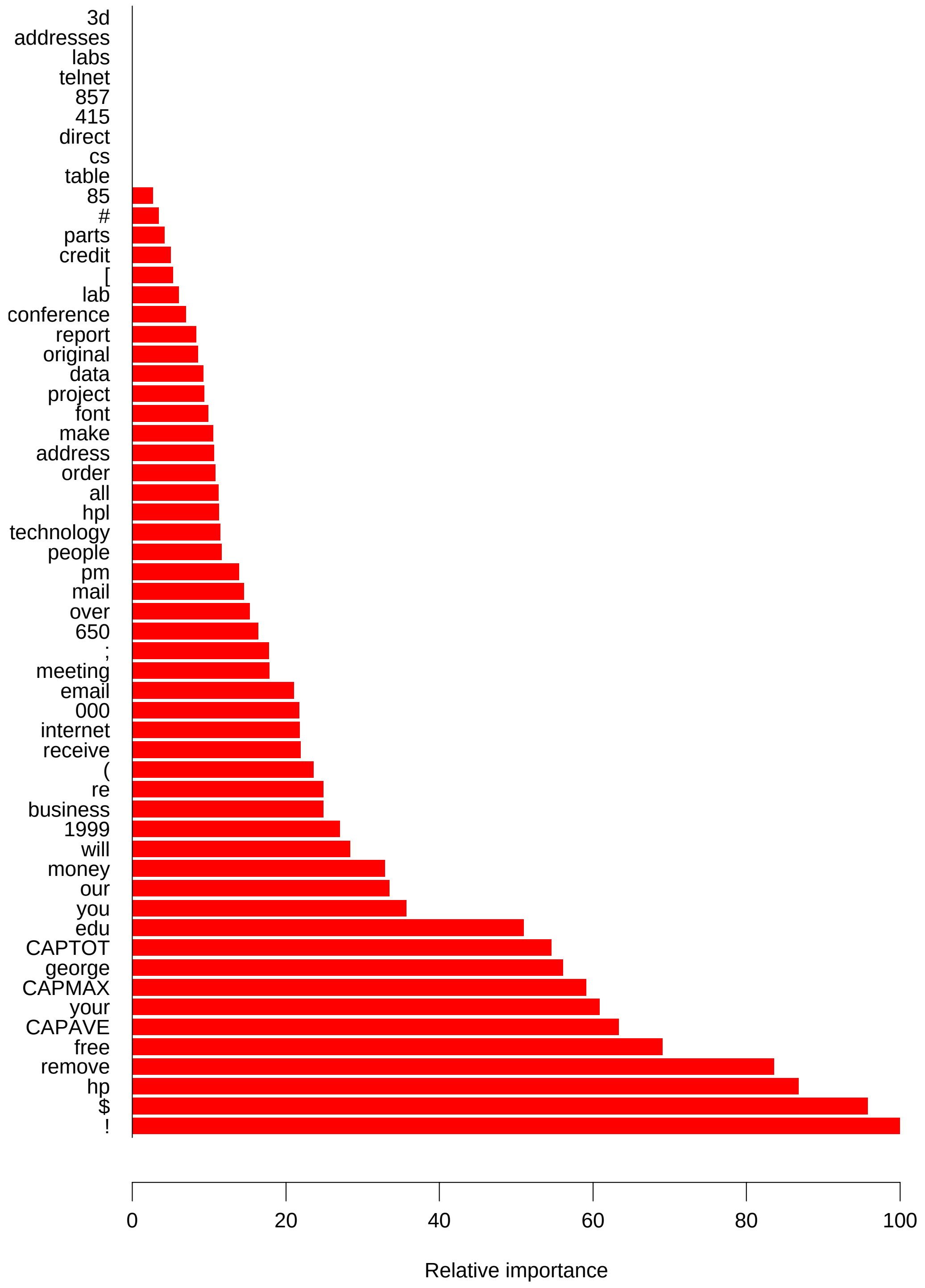
With 3000 training and 1500 test observations, MART fits an additive logistic model

$$f(x) = \log \frac{\Pr(\texttt{spam}|x)}{\Pr(\texttt{email}|x)}$$

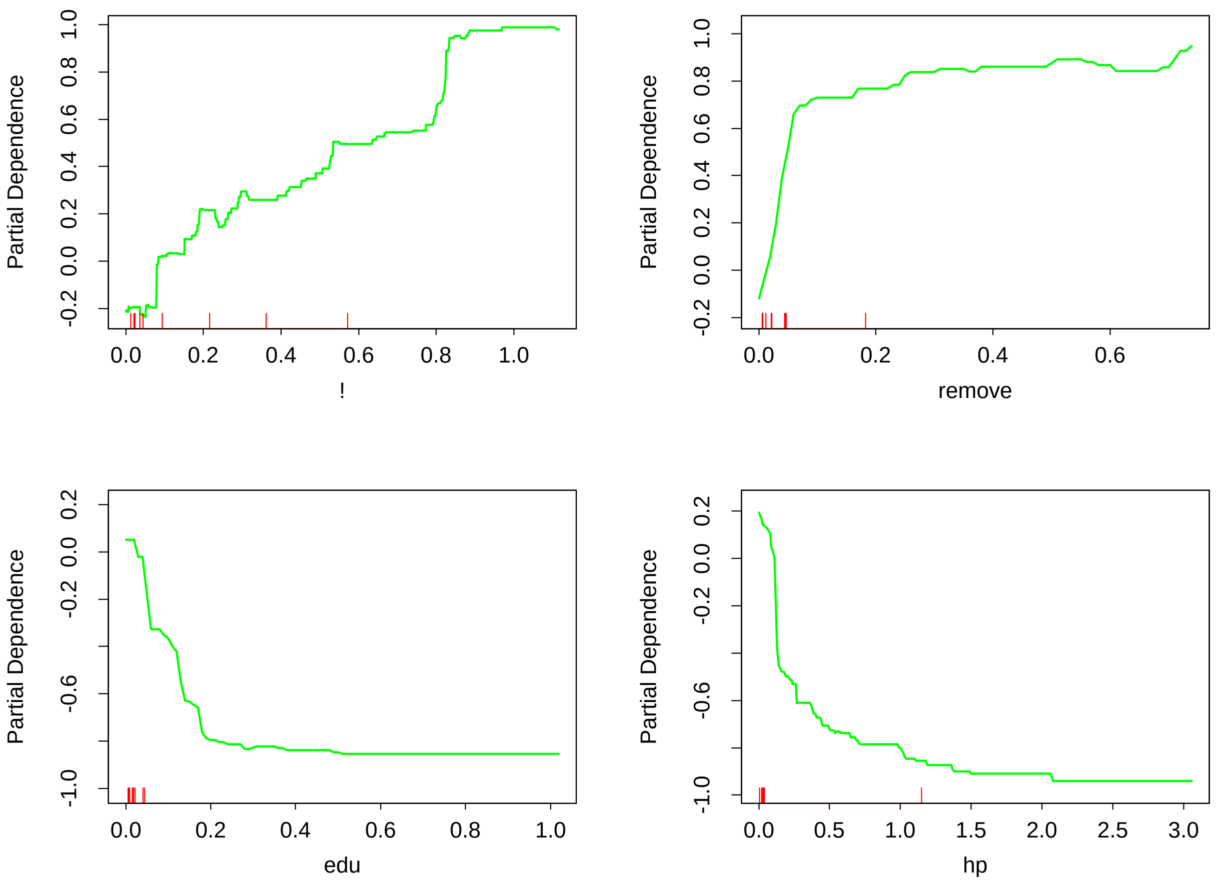
using trees with $J = 6$ terminal-node trees.

MART achieves a test error of 4%, compared to 5.3% for an additive GAM, 5.5% for MARS, and 8.7% for CART.

Spam: Variable Importance

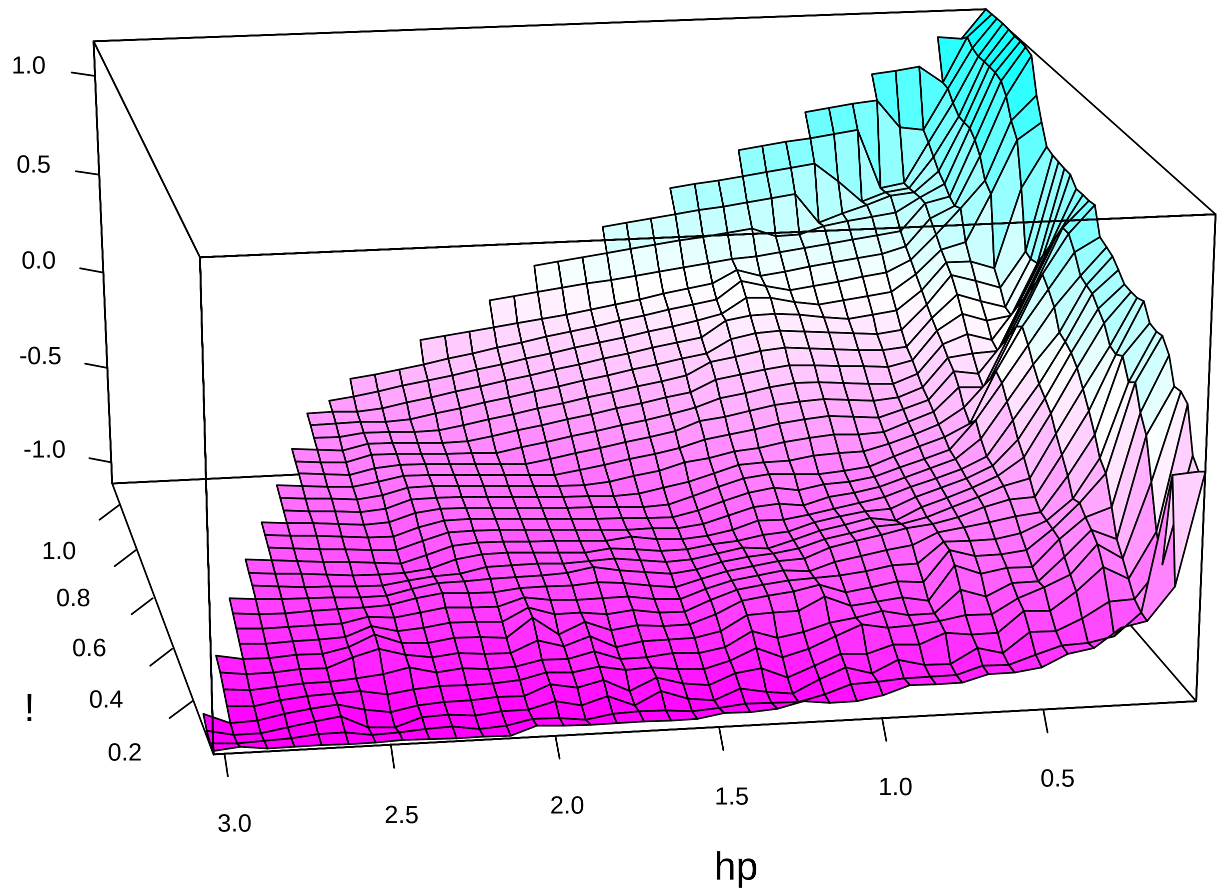


Spam: Partial Dependence



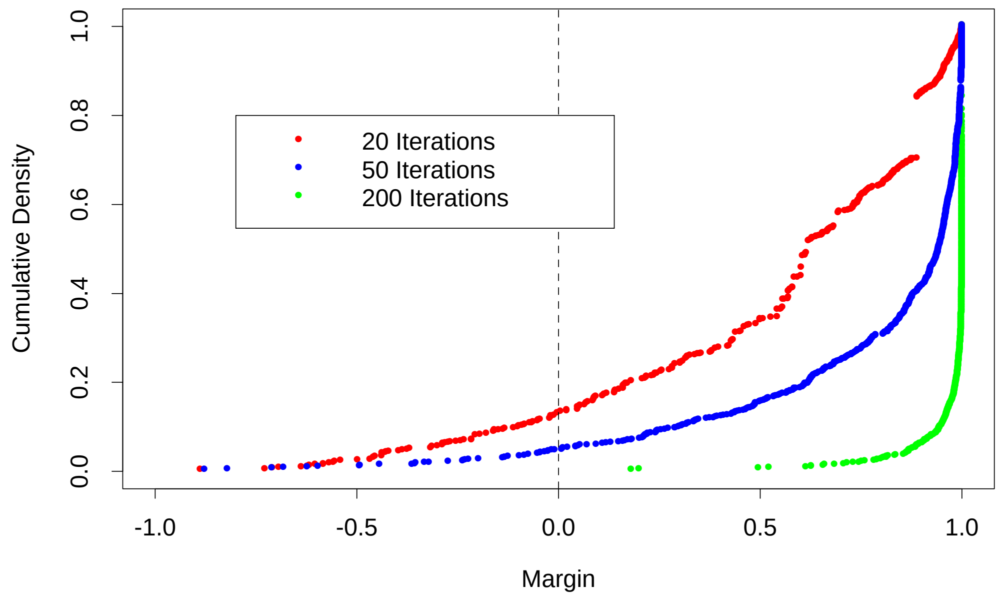
$$\bar{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \frac{1}{N}\sum_{i=1}^{N} f(X_{\mathcal{S}}, x_{i\mathcal{C}})$$

Spam: Partial Dependence



$$\bar{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \frac{1}{N}\sum_{i=1}^{N} f(X_{\mathcal{S}}, x_{i\mathcal{C}})$$

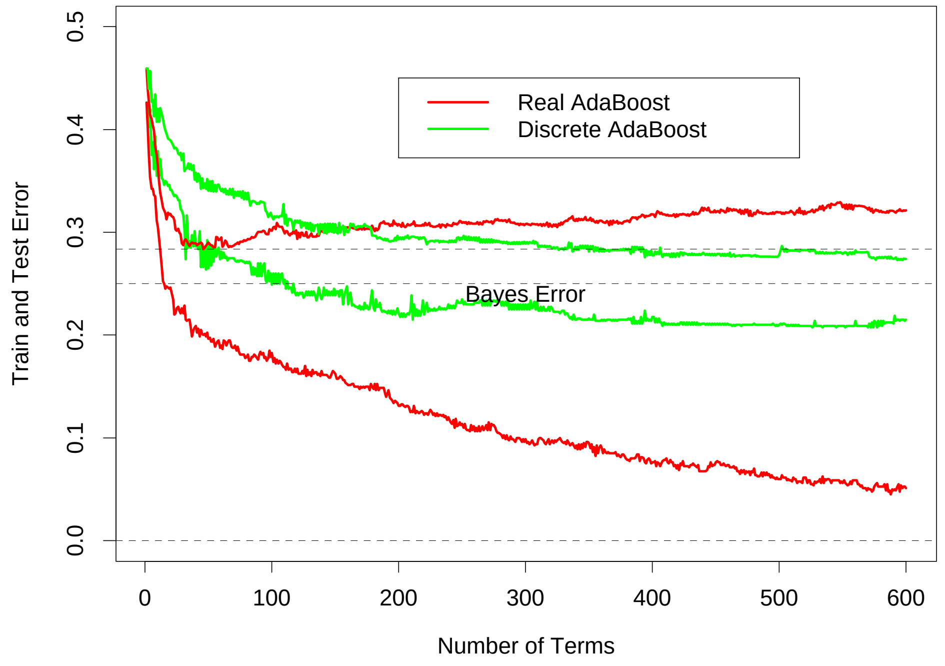
Margins



$$\text{margin}(X) = M(X) = 2\hat{P}_{C(X)} - 1$$

Freund & Schapire (1997): Boosting generalizes because it pushes the training margins well above zero, while keeping the VC dimension under control (also Vapnik, 1996). With $Pr \geq (1-\delta)$

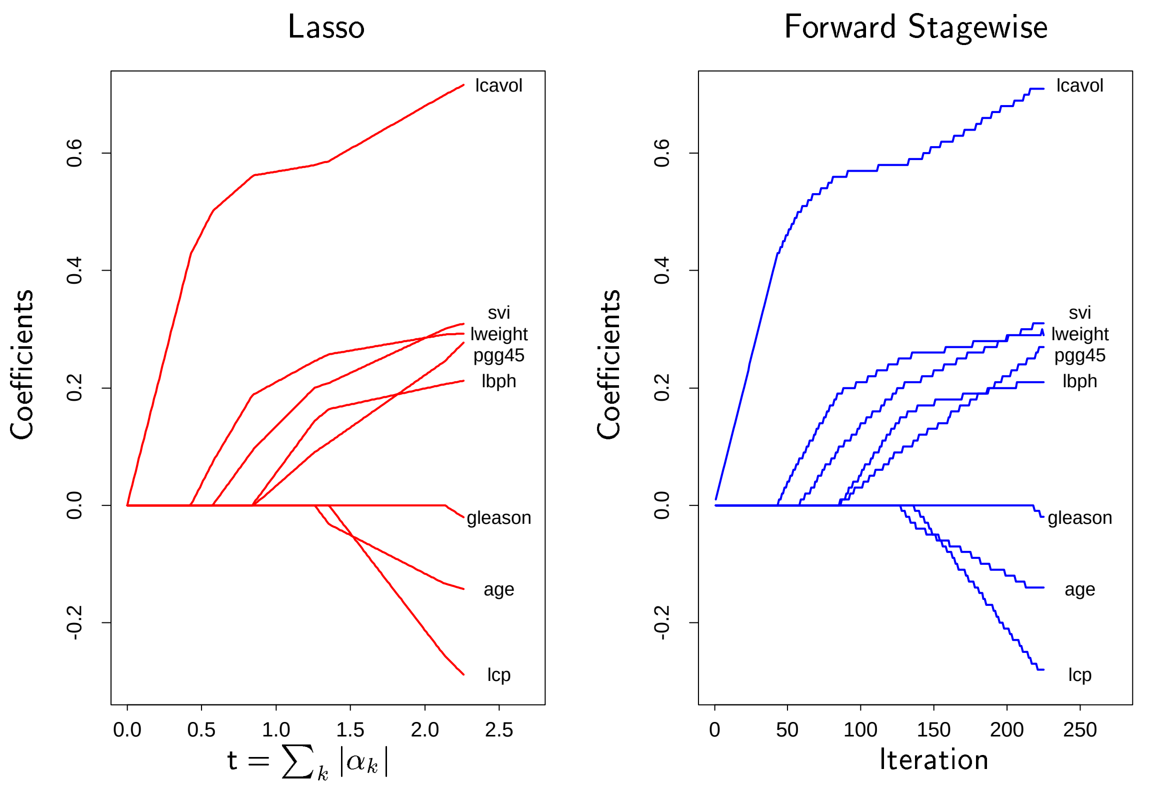
$$P_{Test}(M(X) \leq 0) \quad \leq \quad P_{Train}(M(X) \leq \theta) + O\left(\frac{1}{\sqrt{N}}\left(\frac{\log N \log|\mathcal{H}|}{\theta^2 + \log 1/\delta}\right)^{\frac{1}{2}}\right)$$



How does Boosting avoid overfitting?

- As iterations proceed, impact of change is localized.
- Parameters are not jointly optimized — stagewise estimation slows down the learning process.
- Classifiers are hurt less by overfitting (Cover and Hart, 1967).
- Margin theory of Schapire and Freund, Vapnik? Disputed by Breiman (1997).
- Jury is still out!

Boosting and L_1 Penalized Fitting



- ϵ forward stagewise:(idealized boosting with shrinkage). Given a family of basis functions $h_1(x), \ldots h_M(x)$, and loss function L.
- Model at kth step is $F_k(x) = \sum_m \beta_m^k h_m(x)$.
- At step $k+1$, identify coordinate m with largest $|\partial L/\partial \beta_m|$, and update $\beta_m^{k+1} \leftarrow \beta_m^k + \epsilon$.
- Equivalent to the lasso: $\min L(\beta) + \lambda_k ||\beta||_1$

Summary and Closing Comments

- The introduction of Boosting by Schapire, Freund, and colleagues has brought us an exciting and important set of new ideas.
- Boosting fits additive logistic models, where each component (base learner) is simple. The complexity needed for the base learner depends on the target function.
- Little connection between weighted boosting and bagging; boosting is primarily a bias reduction procedure, while the goal of bagging is variance reduction.
- Boosting can of course overfit; stopping early is a good way to regularize boosting.
- Our view of boosting is stagewise and slow optimization of a loss function. For example, gradient boosting approximates the gradient updates by small trees fit to the empirical gradients.

- Modern versions of boosting use different loss functions, and incorporate shrinkage. Can handle a wide variety of regression modeling scenarios.
- Later on we will compare boosting with support vector machines.

Comparison of Learning Methods

Some characteristics of different learning methods.
Key: 🟢= good, 🟡=fair, and 🔴=poor.

Characteristic	Neural Nets	SVM	CART	GAM	KNN, kernels	MART
Natural handling of data of “mixed” type	🔴	🔴	🟢	🟢	🔴	🟢
Handling of missing values	🔴	🔴	🟢	🔴	🟢	🟢
Robustness to outliers in input space	🔴	🔴	🟢	🟡	🟢	🟢
Insensitive to monotone transformations of inputs	🔴	🔴	🟢	🟢	🔴	🟢
Computational scalability (large N)	🔴	🔴	🟢	🟢	🔴	🟢
Ability to deal with irrelevant inputs	🔴	🔴	🟢	🟡	🔴	🟢
Ability to extract linear combinations of features	🟢	🟢	🔴	🔴	🟡	🟡
Interpretability	🔴	🔴	🟡	🟢	🔴	🟢
Predictive power	🟢	🟢	🔴	🔴	🟢	🟢