Additive Logistic Regression
a Statistical View of Boosting

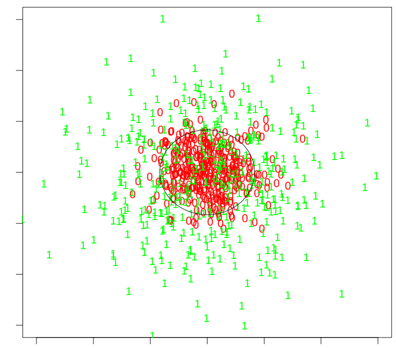
Jerome Friedman, Trevor Hastie, Rob Tibshirani
Stanford University

Thanks to Bogdan Popescu for helpful and very lively discussions on the history of boosting, and for help in preparing that part of this talk

`Email: trevor@stat.stanford.edu`
`Ftp: stat.stanford.edu: pub/hastie`
`WWW: http://www-stat.stanford.edu/~trevor`

These transparencies are available via ftp:
`ftp://stat.stanford.edu/pub/hastie/boost98.ps`

Classification Problem



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

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

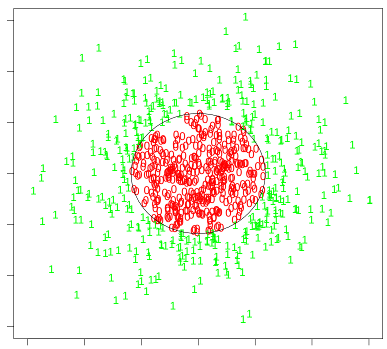
(X, Y) have joint probability distribution $\mathcal{D}$.

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



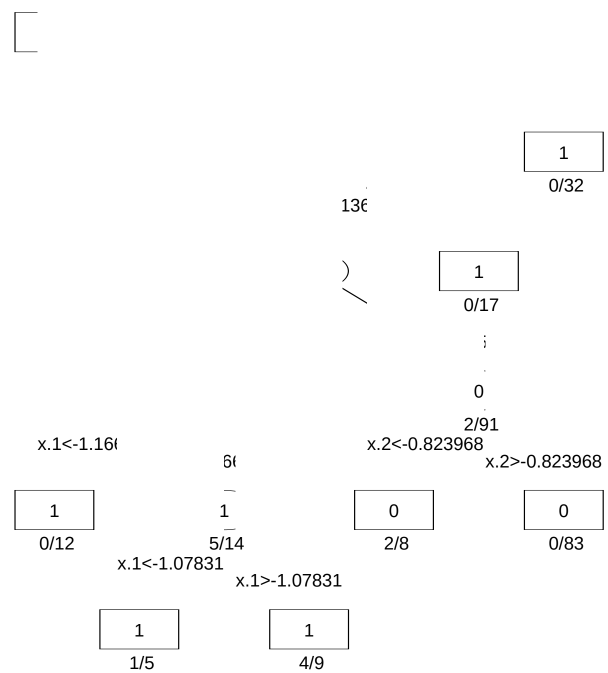
$X \in R^p$ has distribution $\mathcal{D}$.

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

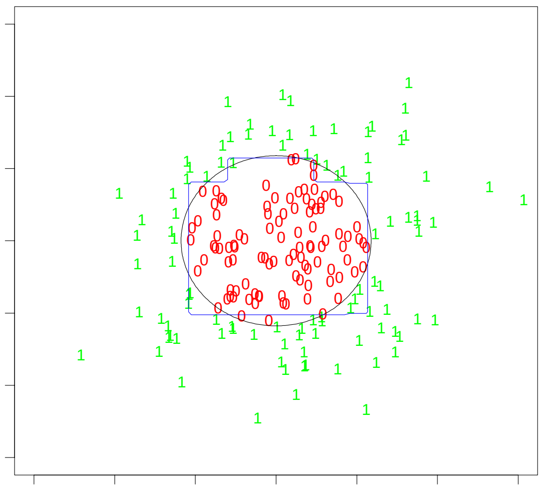
Goal: Based on N training pairs $(X_i,\ Y_i = C(X_i))$ drawn from $\mathcal{D}$ produce a classifier $\hat{C}(X) \in \{0, 1\}$

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

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



Decision Boundary: Tree



When the nested spheres are in R^{10}, CART^{TM} produces a rather noisy and inaccurate rule $\hat{C}(X)$, with error rates around 40%.

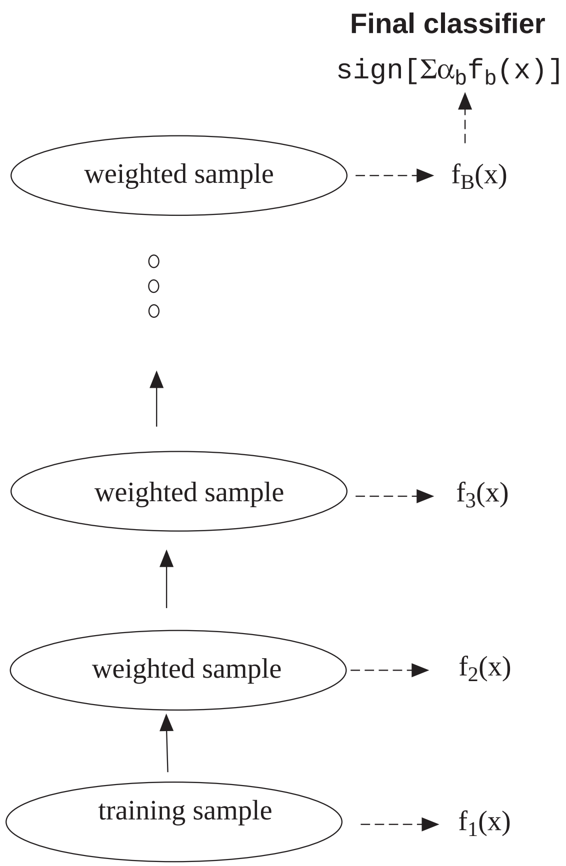
Bagging and Boosting

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.

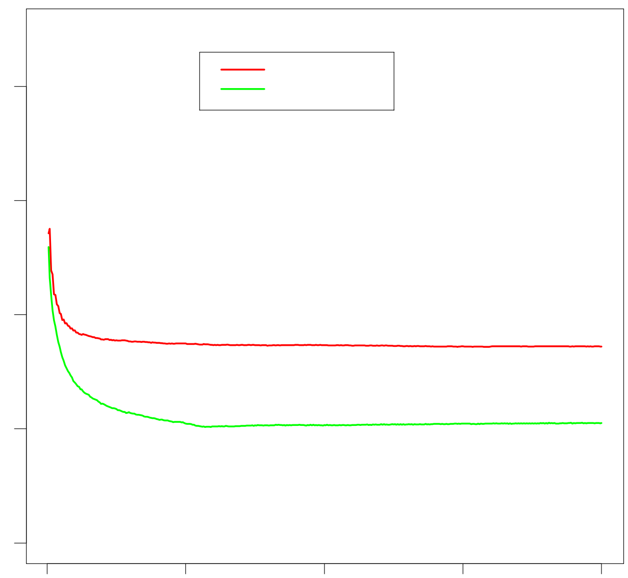
In general Boosting > Bagging > Single Tree.

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



The weighting in boosting can be achieved by weighted importance sampling.

Bagging and Boosting

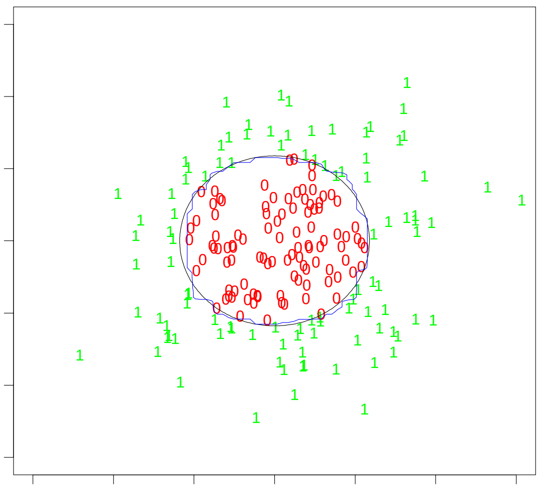


2000 points from Nested Spheres in R^{10}; Bayes error rate is 0%.

Trees are grown Best First without pruning.

Leftmost iteration is a single tree.

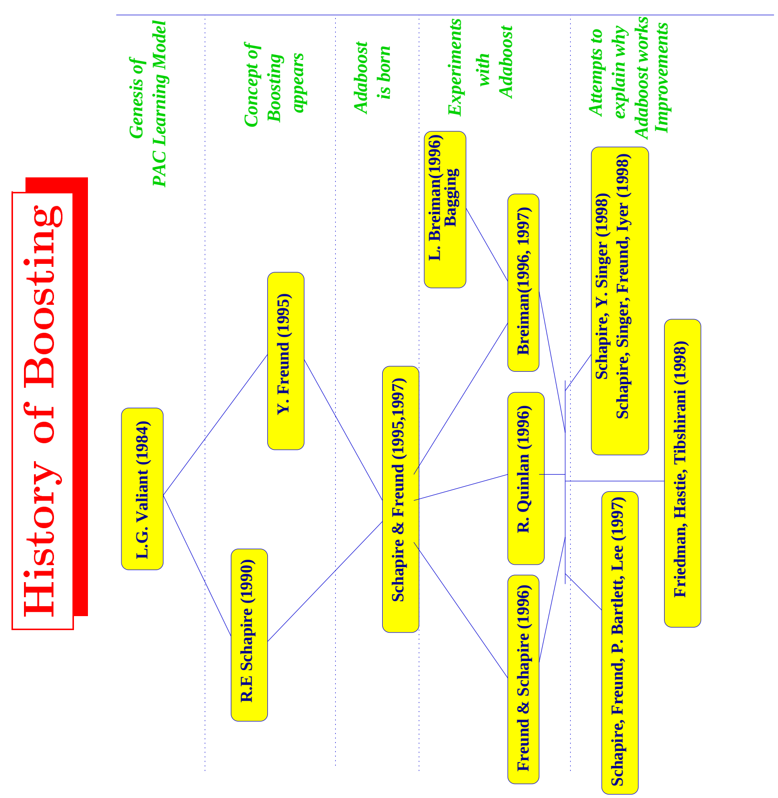
Decision Boundary: Boosting



Bagging and Boosting average many trees, and produce smoother decision boundaries.

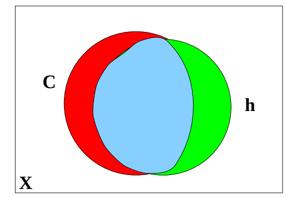
AdaBoost (Freund & Schapire, 1996)

1. Start with weights $w_i = 1/N \ \forall i = 1, \ldots, N$. $y_i \in \{-1, 1\}$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Estimate the weak learner $f_m(x) \in \{-1, 1\}$ from the training data with weights w_i.
   (b) Compute $e_m = E_w[1(y \neq f_m(x)]$, $c_m = \log((1 - e_m)/e_m)$.
   (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 majority weight classifier $C(x) = \text{sign}[\sum_{m=1}^{M} c_m f_m(x)]$.



PAC Learning Model

$X \sim \mathcal{D}$: Instance Space
$C : X \mapsto \{0, 1\}$ Concept $\in \mathcal{C}$
$h : X \mapsto \{0, 1\}$ Hypothesis $\in \mathcal{H}$
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. 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.

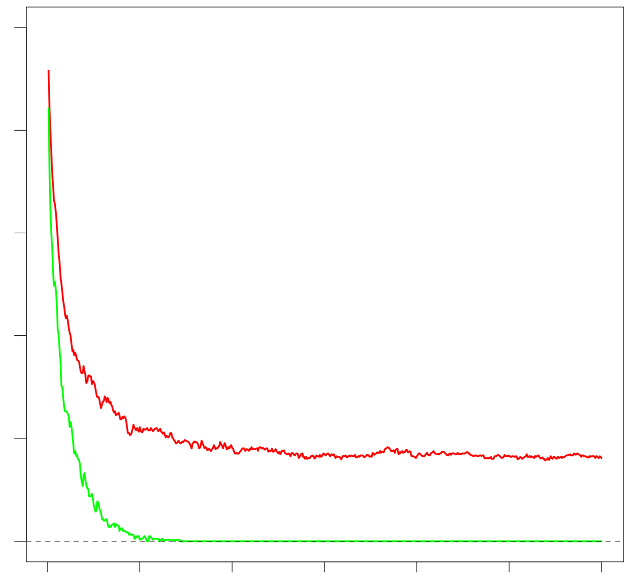
1. L learns h_1 on first N training points.
2. L randomly filters the next batch of training points, extracting $N/2$ points correctly classified by h_1, $N/2$ incorrectly classified, and produces h_2.
3. L builds a third training set of N points for which h_1 and h_2 disagree, and produces h_3.
4. L 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 & Training Error

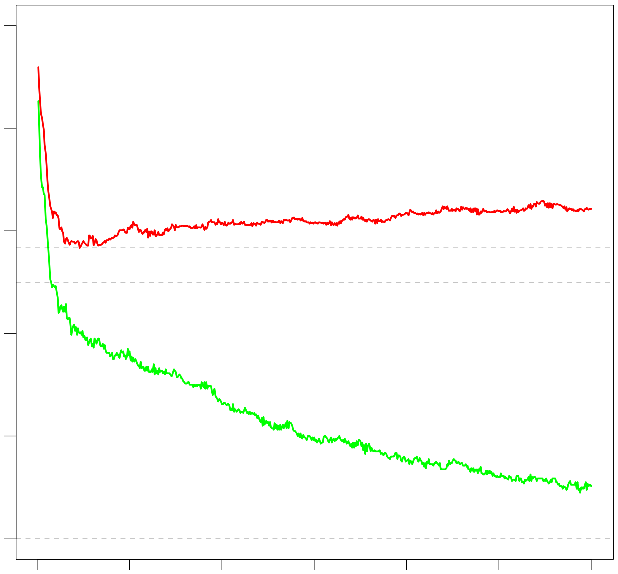
Nested spheres in R^{10} — Bayes error is 0%.



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

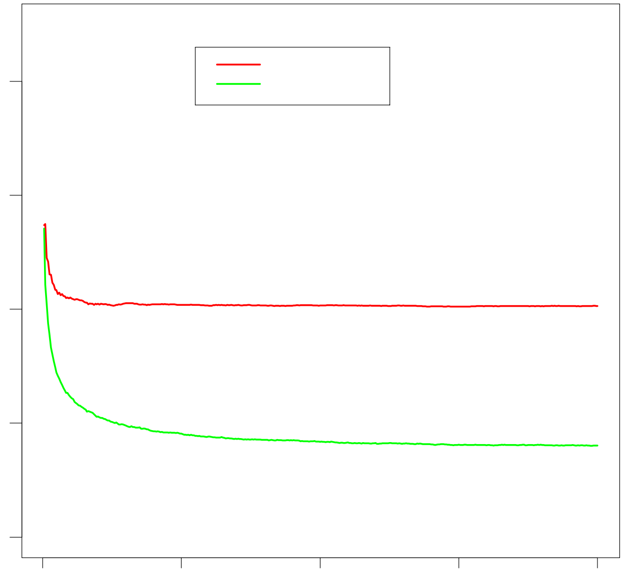
Boosting Noisy Problems

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



Here the test error does increase, but quite slowly.

Bagging and Boosting Smaller Trees

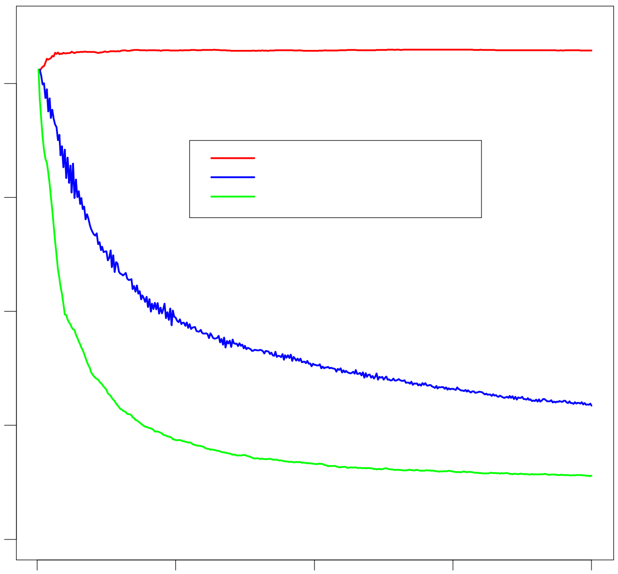


2000 points in R^{10}; Bayes error rate is 0%.

Each tree has 10 terminal nodes, grown best-first.

Bagging-Boosting gap is wider

Bagging and Boosting Stumps



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

Each tree has 2 terminal nodes, grown best-first.

Bagging fails — boosting does best ever!

Prediction games

Results of Freund and Schapire (1996) and Breiman (1998).

Idea:

- Start with fixed learners $f_1(x), f_2(x) \ldots f_M(x)$.
- Play a two person game: player 1 picks observation weights w_i; player 2 picks learner weights c_m (i.e. he will use learner $f_m(x)$ with probability c_m).
- Player 1 tries to make the prediction problem as hard as possible, while Player 2 does the best he can on the weighted problem. We judge difficulty by the approximate "margin", a smooth version of misclassification loss.

- Result: this is a zero-sum game, and the minimax theorem gives the best strategy for each player. Furthermore, AdaBoost converges to this optimal strategy!
- However: link with statistical properties of actual AdaBoost is tenuous:
  1. why minimize hardest weighted problem (makes test error smaller?);
  2. actual AdaBoost does not use a random choice of learner,
  3. actual AdaBoost finds the learners $f_m(x)$.

Stage-wise Additive Modeling

Boosting builds an additive model $F(x) = \sum_{m=1}^{M} f_m(x)$ and then $C(x) = \text{sign}[F(x)]$.

We do things like that in statistics!

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

Traditionally each of the terms $f_m(x)$ is different in nature, and they are fit jointly (i.e. least squares, maximum likelihood)

With Boosting, each term is equivalent in nature, and they are fit in a stagewise fashion.

Simple example: stagewise least-squares? Fix the past $M-1$ functions, and update the Mth using a tree:

$$\min_{f_M \in Tree(x)} E(Y - \sum_{m=1}^{M-1} f_m(x) - f_M(x))^2$$

Boosting and Additive Models

- Discrete AdaBoost builds an additive model

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

by stage-wise optimization of

$$J(F(x)) = Ee^{-y(F(x))}$$

- Given an imperfect $F_{M-1}(x)$, the updates in Discrete AdaBoost correspond to a Newton step towards minimizing

$$J(F_{M-1}(x) + c_M f_M(x)) = Ee^{-y(F_{M-1}(x)+c_M f_M(x))}$$

over $f_M(x) \in \{-1, 1\}$, with step length c_M.

- $Ee^{-y(F(x))}$ is minimized at

$$F(x) = \frac{1}{2} \log \frac{P(y = 1|x)}{P(y = -1|x)}$$

Hence Adaboost is fitting an additive logistic regression model.

Details

f_M:

$$f_M(x) = \arg \min_{g(x) \in \{-1,1\}} E_w(y - g(x))^2$$

with weights $w(x, y) = e^{-yF_{M-1}(x)}$.

c_M:

$$\begin{aligned} c_M &= \arg\min_c E_w e^{-cyf_M(x)} \\ &= \frac{1}{2} \log \frac{1-e}{e} \end{aligned}$$

with $e = E_w[1_{[y \neq f_M(x)]}]$.

- Empirical version: at each stage, $f_M(x)$ is estimated by the classification at the terminal nodes of a tree, grown to appropriately weighted versions of the training data.

- At the Mth stage of the Discrete AdaBoost iterations, the weights are such that f_{M-1} has weighted training error $= 50\%$.

Real AdaBoost

1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$. $y_i \in \{-1, 1\}$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the 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 $\text{sign}[\sum_{m=1}^{M} f_m(x)]$

Real AdaBoost

- Real AdaBoost also builds an additive logistic regression model:

$$\log \frac{P(y = 1|x)}{P(y = -1|x)} = \sum_{m=1}^{M} f_m(x)$$

  by stage-wise optimization of $J(F(x)) = Ee^{-y(F(x))}$.

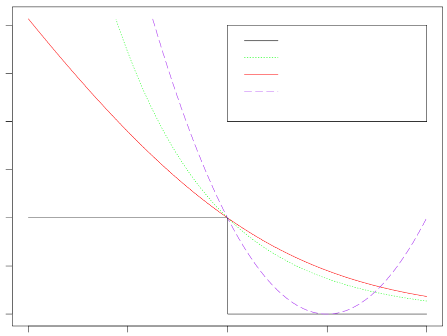
- Given an imperfect $F_{M-1}(x)$, Real AdaBoost minimizes $J(F_{M-1}(x) + f_M(x)) = Ee^{-y(F_{M-1}(x) + f_M(x))}$ over $f_M(x) \in R$, with solution

$$f_M(x) = \frac{1}{2} \log \frac{P_w(y = 1)}{P_w(y = -1)}$$

  where the weights $w(x, y) = e^{-yF_{M-1}(x)}$.

- Empirical version: at each stage, $P_w(\cdot|x)$ is estimated by averages at the terminal nodes of a tree, grown to appropriately weighted versions of the training data.

Why $J(F(x)) = Ee^{-yF(x)}$?



- $e^{-yF(x)}$ is a monotone, smooth upper bound on misclassification loss at x.
- $J(F)$ is an expected χ statistic at its minimum, and equivalent to the binomial log-likelihood to second order.
- Stage-wise binomial maximum-likelihood estimation of additive models based on trees works as least as well.

Stagewise Maximum Likelihood

Consider the model

$$F_M(x) = \log \frac{P(y^* = 1|x)}{P(y^* = 0|x)} = \sum_{m=1}^{M} f_m(x)$$

or

$$P(y^* = 1|x) = p(x) = \frac{e^{F(x)}}{1 + e^{F(x)}}$$

The binomial log-likelihood is

$$\begin{aligned} \ell(F(x)) &= E[y^* \log(p(x)) + (1 - y^*) \log(1 - p(x))] \\ &= E[y^* F(x) - \log(1 + e^F(x)) \end{aligned}$$

Stagewise Maximum Likelihood: Given an imperfect $F_{M-1}(x)$, maximize $\ell(F_{M-1}(x) + f_M(x))$ over $f_M(x) \in R$

The LogitBoost algorithm takes a single Newton-Step at each stage.

LogitBoost

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. i.e. a weighted tree

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

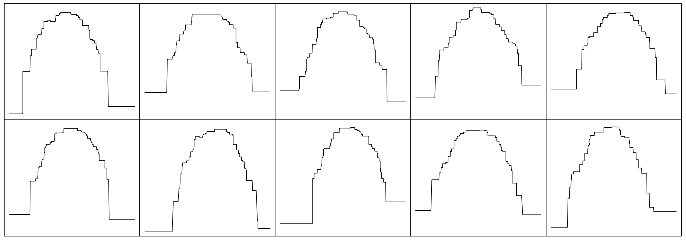
We also have a natural generalization of LogitBoost for multiple classes.

Additive Logistic Trees

Best-first tree growing allows us to limit the size of each tree, and hence the interaction order.

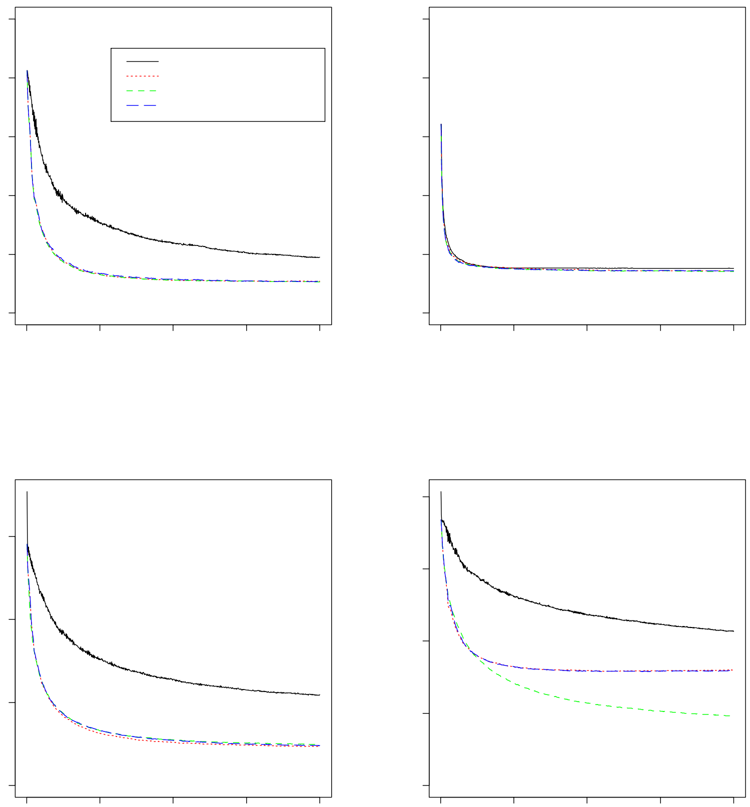
By collecting terms, we get

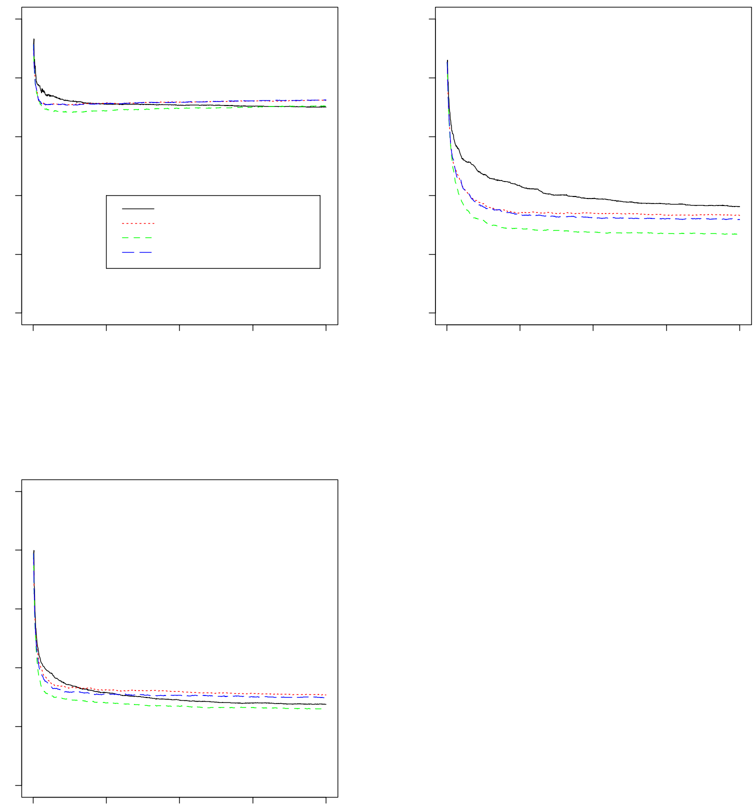
$$F(x) = \sum_j f_j(x_j) + \sum_{j,k} f_{jk}(x_j, x_k) + \sum_{j,k,l} f_{jkl}(x_j, x_k, x_l) + \ldots$$



Coordinate functions for Additive Stumps Model.

Boosting uses stage-wise optimization, as opposed to joint optimization (full least squares, backfitting,$\cdots$).





Large Real Example: Satimage

Method	Terminal Nodes	Iterations 20	50	100	200
Satimage	CART error = .148				
LogitBoost	2	.140	.120	.112	.102
Real AdaBoost	2	.148	.126	.117	.119
Gentle AdaBoost	2	.148	.129	.119	.119
Discrete AdaBoost	2	.174	.156	.140	.128
LogitBoost	8	.096	.095	.092	.088
Real AdaBoost	8	.105	.102	.092	.091
Gentle AdaBoost	8	.106	.103	.095	.089
Discrete AdaBoost	8	.122	.107	.100	.099

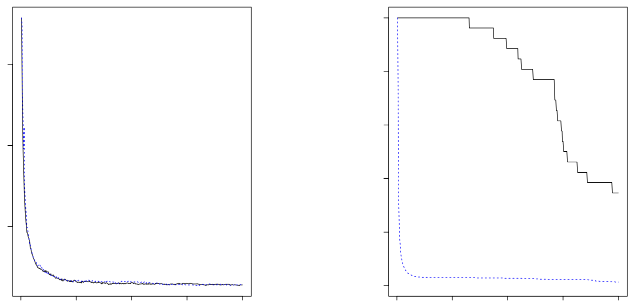
4435 training, 2000 test, 36 features, 6 classes

Large Real Example: Letter

Method	Terminal Nodes	Iterations 20	50	100	200	Fraction
Letter	CART error = .124					
LogitBoost	2	.250	.182	.159	.145	.06
Real AdaBoost	2	.244	.181	.160	.150	.12
Gentle AdaBoost	2	.246	.187	.157	.145	.14
Discrete AdaBoost	2	.310	.226	.196	.185	.18
LogitBoost	8	.075	.047	.036	.033	.03
Real AdaBoost	8	.068	.041	.033	.032	.03
Gentle AdaBoost	8	.068	.040	.030	.028	.03
Discrete AdaBoost	8	.080	.045	.035	.029	.03

16000 training, 4000 test, 16 features, 26 classes

Weight Trimming

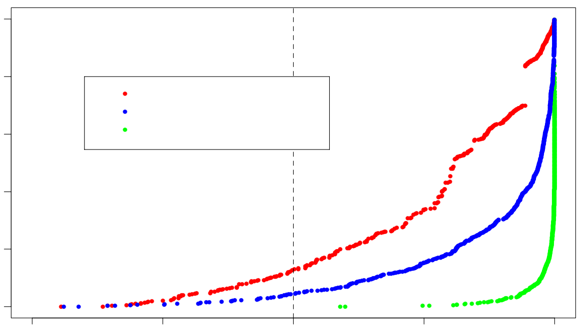


- At each iteration, observations with $w_i < t(\beta)$ are not used for training. $t(\beta)$ is βth quantile of weight distribution, and $\beta \in [0.01, 0.1]$. Works better for LogitBoost:
  - LogitBoost has weights $w_i = p_i(1 - p_i)$ which are large near the decision boundary.
  - AdaBoost has weights $w_i = e^{-y_i F_M(x_i)}$ (recall $y_i \in \{-1, 1\}$). Large for misclassified points.
- For multiple-class procedures, if the class-k logit $F_{mk} > 15 + \log(N)$, training stops for that class.

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.
- The distinction becomes blurred when weighting is achieved (in boosting) by importance sampling.

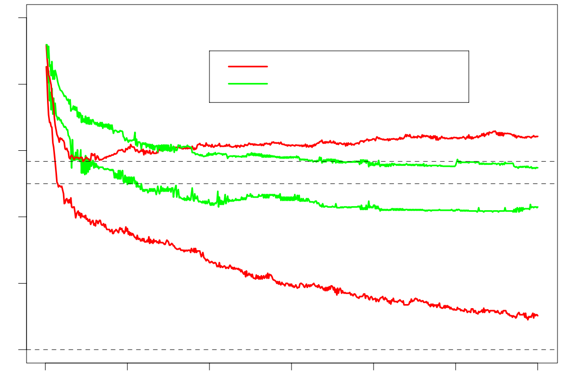
Margins



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

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!