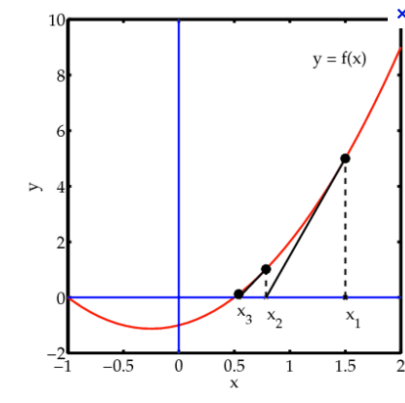


# Committee Machines: Boosting Trees



Rachael Hageman Blair

# Outline

- Recap Adaboost
- Trees
- Boosting Trees
- Gradient Boosting motivation
- Steepest Descent
- Gradient Boosting algorithm
- Improving Gradient Boosting
- Interpretation (brief)
- Summary

# AdaBoost.M1

- At each boosting step, weights are applied  $w_1, w_2, \dots, w_N$ , to each of the training observations  $(x_i, y_i)$ ,  $i = 1, 2, \dots, N$ .
- At the initial step,  $w_i = 1/N$ ,  $\forall i$ .
- At each successive iteration,  $m = 2, 3, \dots, M$ , the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations.
- At step  $m$ , those observations that were misclassified by the classifier  $G_{m-1}(x)$  induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly.

# AdaBoost.M1

---

**Algorithm 10.1** *AdaBoost.M1*.

---

1. Initialize the observation weights  $w_i = 1/N$ ,  $i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - (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) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$ ,  $i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
- 

AdaBoost.M1 aka “Discrete Adaboost”, because returns a discrete class label.  
Real AdaBoost – specifically for real valued predictions.  
Others include: gentle adaboost, modest adaboost.

# Trees

- Recall: Regression and classification trees partition the model space into disjoint regions:  $R_1, R_2, \dots, R_J$ , which are represented as the terminal nodes of the tree.
- A constant  $\gamma_j$  is assigned to each region, such that,

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

- Thus, a tree can be formally expressed as:

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

where  $\Theta = \{R_j, \gamma_j\}_1^J$ .

- Solution requires numerical optimization:

$$\Theta = \arg \min_{\Theta} \sum_{j=1}^J \sum_{x_i \in R_j} L(y_i, \gamma_j).$$

# Trees

2-stage solution:

1. Find  $\gamma_j$  given  $R_j$ : trivial,  $\hat{\gamma}_j = \bar{y}_j$  by the modeling assumptions.
2. Find  $R_j$ : more complicated, use greedy, top-down recursive partitioning algorithm.

Recall: in two dimensions, splitting half planes and enumerating all pairs of split variables and split points.

# Boosting Trees

- The Boosted Tree:

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

Derived in a forward stagewise manner.

- At each step, the minimization problem is solved:

$$\hat{\Theta}_m = \arg \min_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)), \text{ ★}$$

over regions and constants  $\hat{\Theta}_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$  of the next tree, given the current model  $f_{m-1}(x)$ .

- As before, given the regions finding the optimal constants in each region is straightforward:

$$\hat{\gamma}_{jm} = \arg \min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}),$$

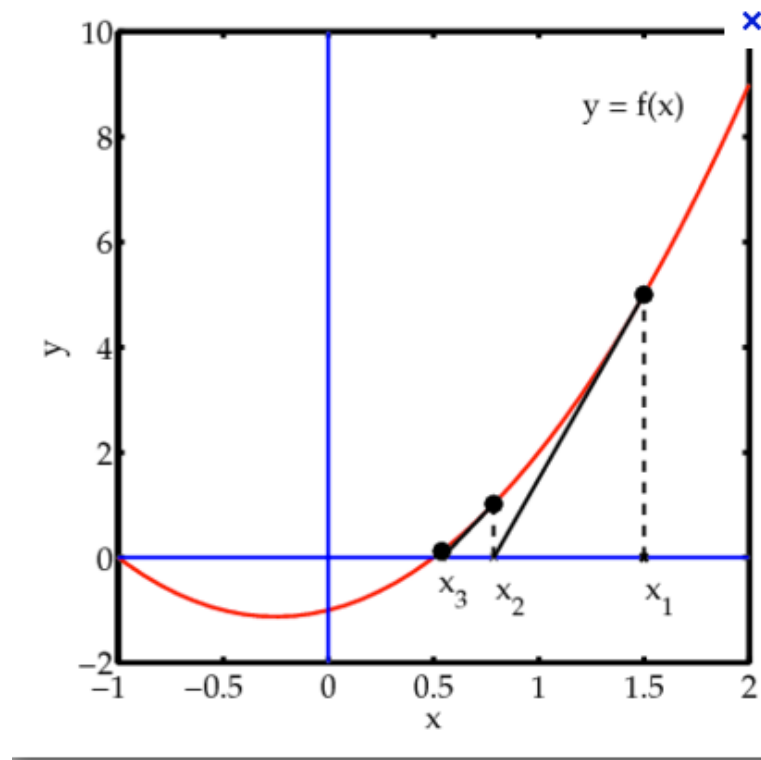
# Gradient Boosting

- General boosting algorithm that works well with a variety of different loss functions.
- Gradient boosting builds additive tree models.
- Tree size is the parameter that determines the order of interaction.
- Gradient boosting inherits all the good features of trees (variable selection, missing data, and mixed predictors), and improves on the weak features, such as prediction performance.



# Gradient Boosting

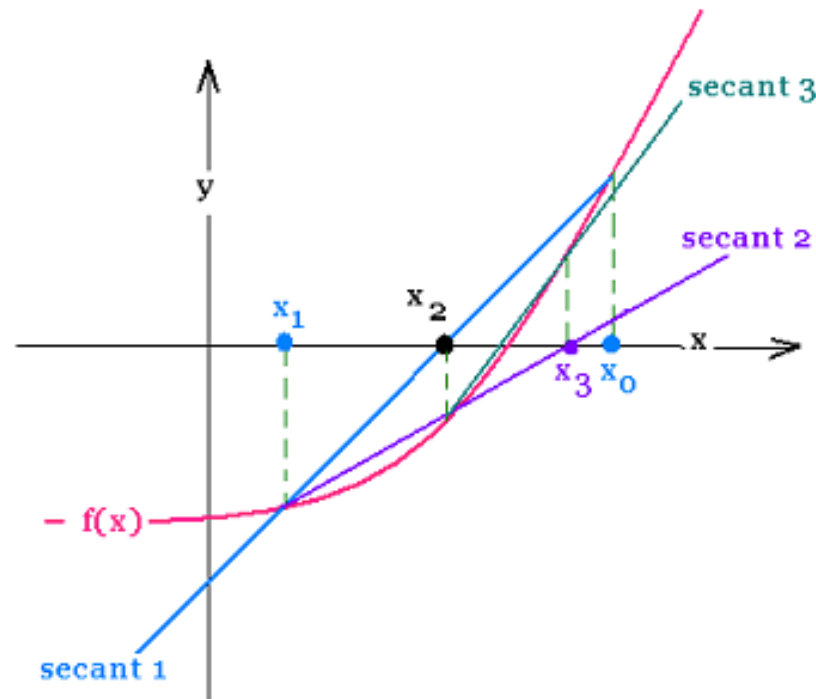
- What if we do not have a special case, and have no analytical solution - > optimization.
- Newton's method – find a zero



# Gradient Boosting

## What if derivatives are unavailable?

- In many practical application  $f'(x)$  is not given by a formula
- $f'(x)$  may not exist over the domain.
- Quasi-Newton methods – rely on  $f(x)$  only, to approximate the solution. In the simplest case, the secant method.



# Gradient Boosting

- Quasi-Newton's methods – find a minimum, no requirement of hessian computation approximated by gradients.
- Consider a general loss:

$$L(f) = \sum_{i=1}^N L(y_i, f(x_i)).$$

the goal is to minimize the loss with respect to  $f(x)$ .

Constrained to be the sum of trees



# Gradient Boosting

- Let the values of the approximating function  $f$  at each of the data points  $x_i$  be denoted as:

$$\mathbf{f} = \{f(x_1), f(x_2), \dots, f(x_N)\}.$$

- The numerical optimization problem:

$$\hat{\mathbf{f}} = \arg \min_{\mathbf{f}} L(\mathbf{f}).$$

where the “parameters” are  $\hat{\mathbf{f}} \in \Re^N$ .

- The solution can be expressed as a sum of component vectors:

$$\mathbf{f}_M = \sum_{m=0}^M \mathbf{h}_m, \quad \mathbf{h}_m \in \Re^N,$$



where  $\mathbf{f}_0 = \mathbf{h}_0$  is an initial guess and each successive  $\mathbf{f}_m$  is induced based on the current parameter vector  $\mathbf{f}_{m-1}$ .

# Steepest Descent

- The solution is obtained via an iterative method, where the approximated function is updated repeatedly until convergence to the minimum. Updates are called “steps”.
- How long should we step toward the minimum? is not a trivial question.
- What direction should we step? is also not a trivial question.
- The method of steepest descent steps as far as possible in the steepest direction.
- The negated gradient is the steepest downhill direction.

# Steepest Descent

- The steepest descent chooses  $\mathbf{h}_m = -\rho_m \mathbf{g}_m$  where  $\rho_m$  is a scalar and  $\mathbf{g}_m \in \Re^N$  is the gradient of  $L(\mathbf{f})$  evaluated at  $\mathbf{f} = \mathbf{f}_{m-1}$ .

- The components of the gradient are:

$$g_{im} = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)}$$

and the step length  $\rho_m$  is the solution to:

$$\rho_m = \arg \min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m).$$

- The update:  $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$ .

# Gradient Boosting

- Steepest descent would be ideal if only concerned with minimizing the loss:

$$L(f) = \sum_{i=1}^N L(y_i, f(x_i)).$$

However, the gradient is only defined at training points  $x_i$  therefore, the approach may not be accurate to new data. The ultimate goal is to generalize  $f_M(x)$  to new data.

- A resolution is to induce a tree  $\{T(x_i; \Theta_m)\}$  as close as possible to the negative gradient,


$$\Theta_m = \arg \min_{\Theta} \sum_{i=1}^N \left( -g_{im} - T(x_i; \Theta) \right)^2$$

Note that the resulting regions will not be the same, but they will be close. The corresponding  $\hat{\gamma}_{jm} = \arg \min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$ ,

# Gradient Boosting

**TABLE 10.2.** Gradients for commonly used loss functions.

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$ y_i - f(x_i) $	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $ y_i - f(x_i)  \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $ y_i - f(x_i)  > \delta_m$ where $\delta_m = \alpha \text{th-quantile}\{ y_i - f(x_i) \}$
Classification	Deviance	$k$ th component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$



K least squares trees are constructed at each iteration.  
Each tree is fit to its corresponding gradient vector.  
\*Only one tree is needed for binary classification.



# Gradient Boosting (MART)

---

**Algorithm 10.3** *Gradient Tree Boosting Algorithm.*

---

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

2. For  $m = 1$  to  $M$ :

(a) For  $i = 1, 2, \dots, N$  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  $R_{jm}$ ,  $j = 1, 2, \dots, J_m$ .

(c) For  $j = 1, 2, \dots, J_m$  compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(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

- Tree size – one approach, decide independently. Does not work well in practice for the boosting.
- Alternative strategy: require all of the trees to be the same size  $J_m = J, \forall m$ , where  $J_m$  is a  $j$ -terminal node tree.
- $J$  is a meta-parameter.

*What is the effect of different size  $J$ ?*

## Tree Size

- Interaction level is limited to tree size and this constraint carries over to boosted tree.

Example:

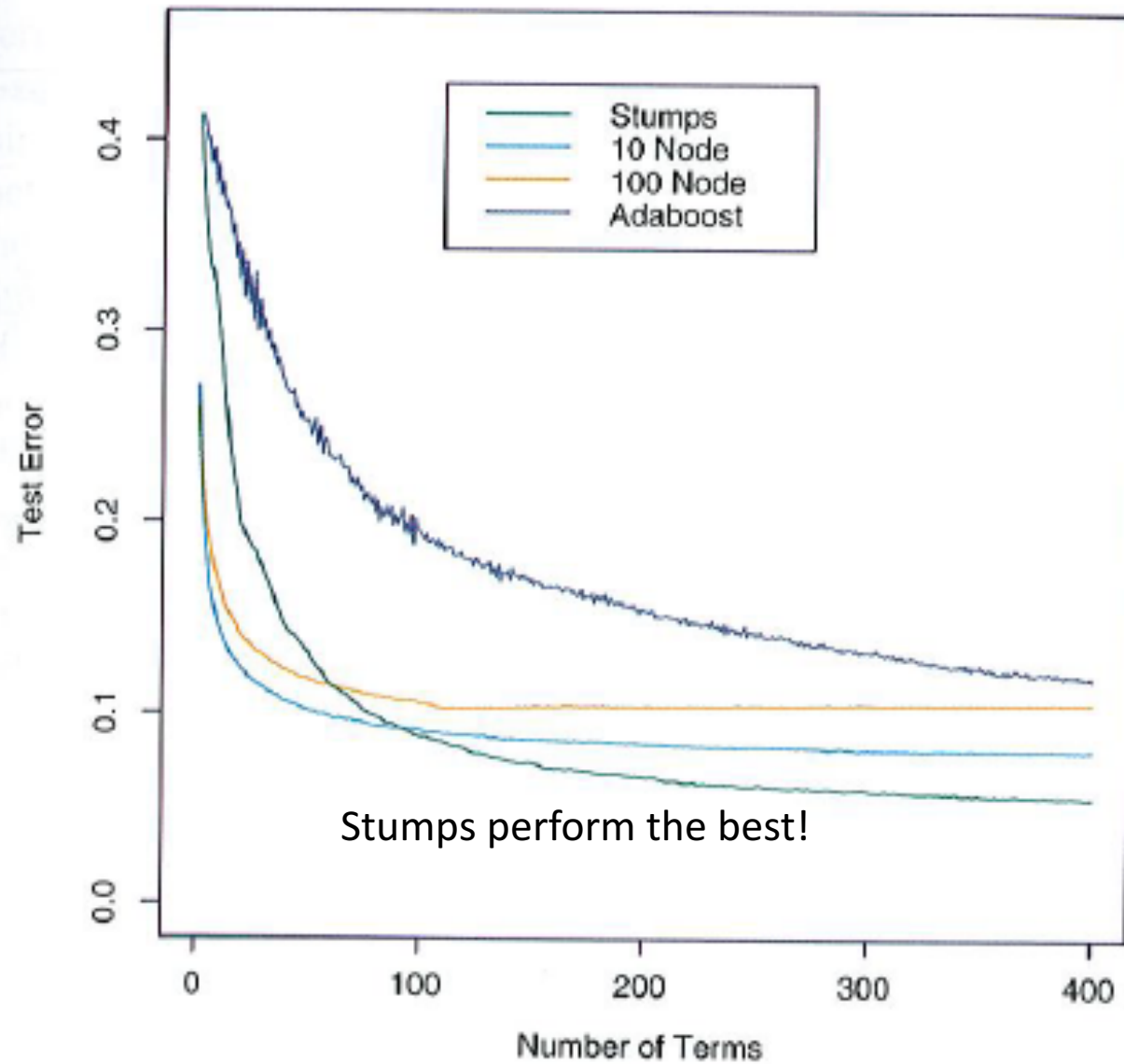
$J = 2 \sim$  produces a boosted model with only main effects.

$J = 3 \sim$  2 variable interactions permitted.

- The value of  $J$  should reflect knowledge about the number of interactions (usually unknown, but most often low).

Rule of thumb:  $J=2$  too small,  $J>10$  too big, between 4 and 10 is good.

# Tree Size



# Shrinkage

- The second consideration, how many trees to include in the boosting.
- Same issues wrt overfitting.
- Strategy: model prediction risk (error) on a validation sample.  
Find the  $M^*$  that minimizes the predicted risk.

# Shrinkage

---

**Algorithm 10.3** *Gradient Tree Boosting Algorithm.*

---

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

2. For  $m = 1$  to  $M$ :

(a) For  $i = 1, 2, \dots, N$  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  $R_{jm}$ ,  $j = 1, 2, \dots, J_m$ .

(c) For  $j = 1, 2, \dots, J_m$  compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(d) Update  $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ .

3. Output  $\hat{f}(x) = f_M(x)$ .


$$f_m(x) = f_{m-1}(x) + v \cdot \sum_{j=1}^J \gamma_{jm} I(x \in R_{jm})$$

# Shrinkage

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^J \gamma_{jm} I(x \in R_{jm})$$

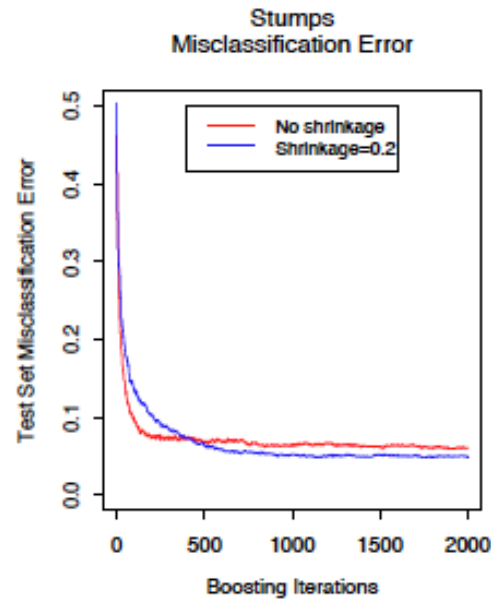
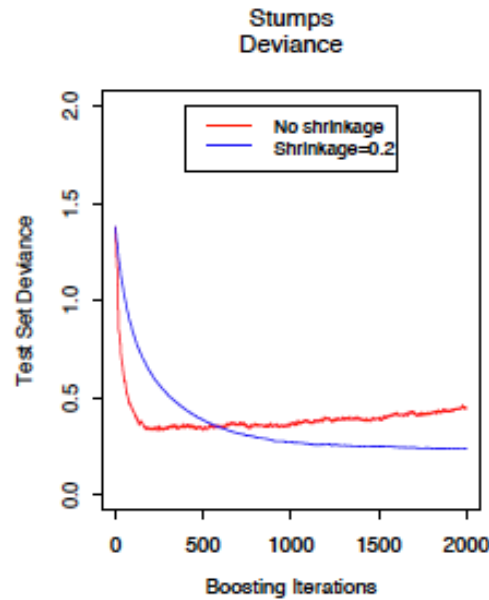
- Now we have another complexity parameter that also controls prediction risk.
- Inverse relationship between  $\nu$  and  $M$ .
- Empirically, it has been shown that smaller values of  $\nu$  favor better test error and require corresponding larger values of  $M$ .

Another possibility.... Subsampling.

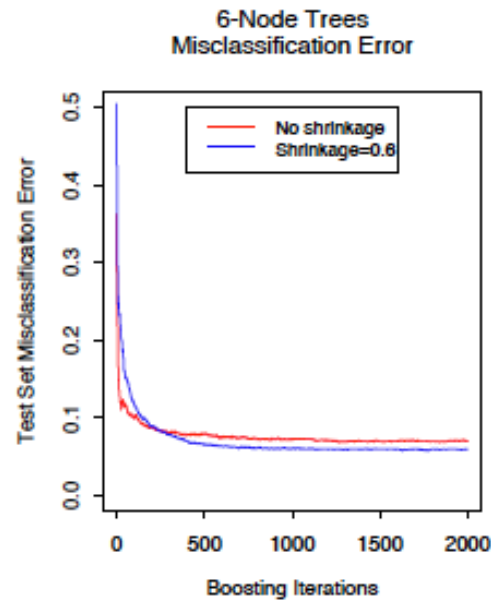
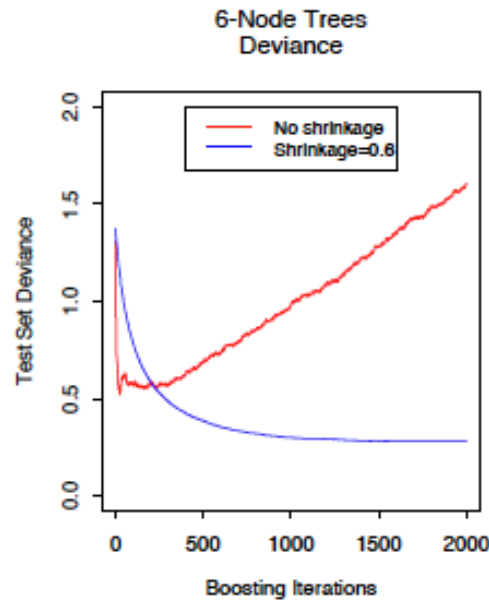
## Deviance

## Misclassification Rate

Stumps



6 Node Tree



Red: Shrinkage  
Blue: No Shrinkage



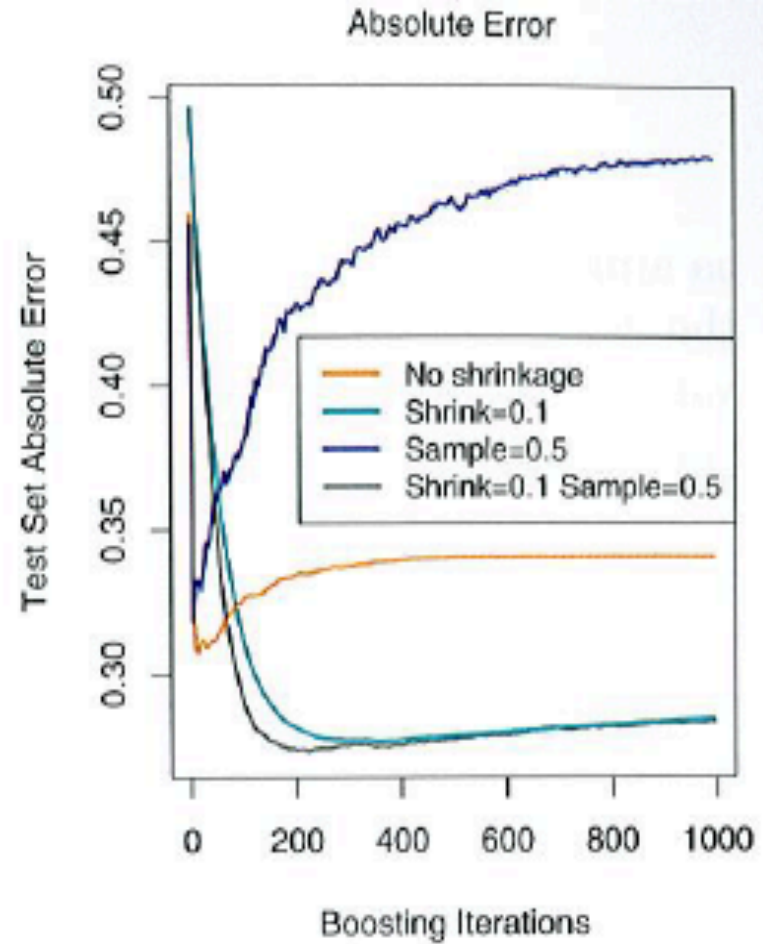
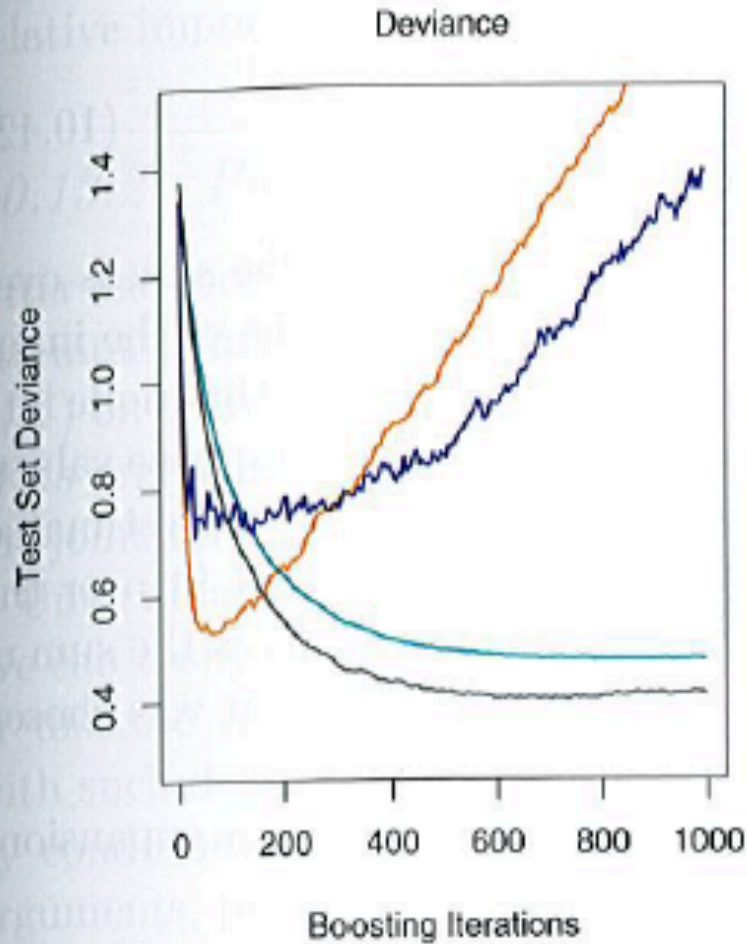
## Number of Trees

**Stochastic Gradient Boosting** ~ at each iteration, we sample a fraction of the training data  $\eta$  without replacement. Grow the next tree using the subsample.

- Reduction in computation
- Often produces a more accurate model when used with shrinkage.
- We are up to four complexity parameters:  $J, M, \nu$ , and  $\eta$ .

# Number of Trees

4-Node Trees



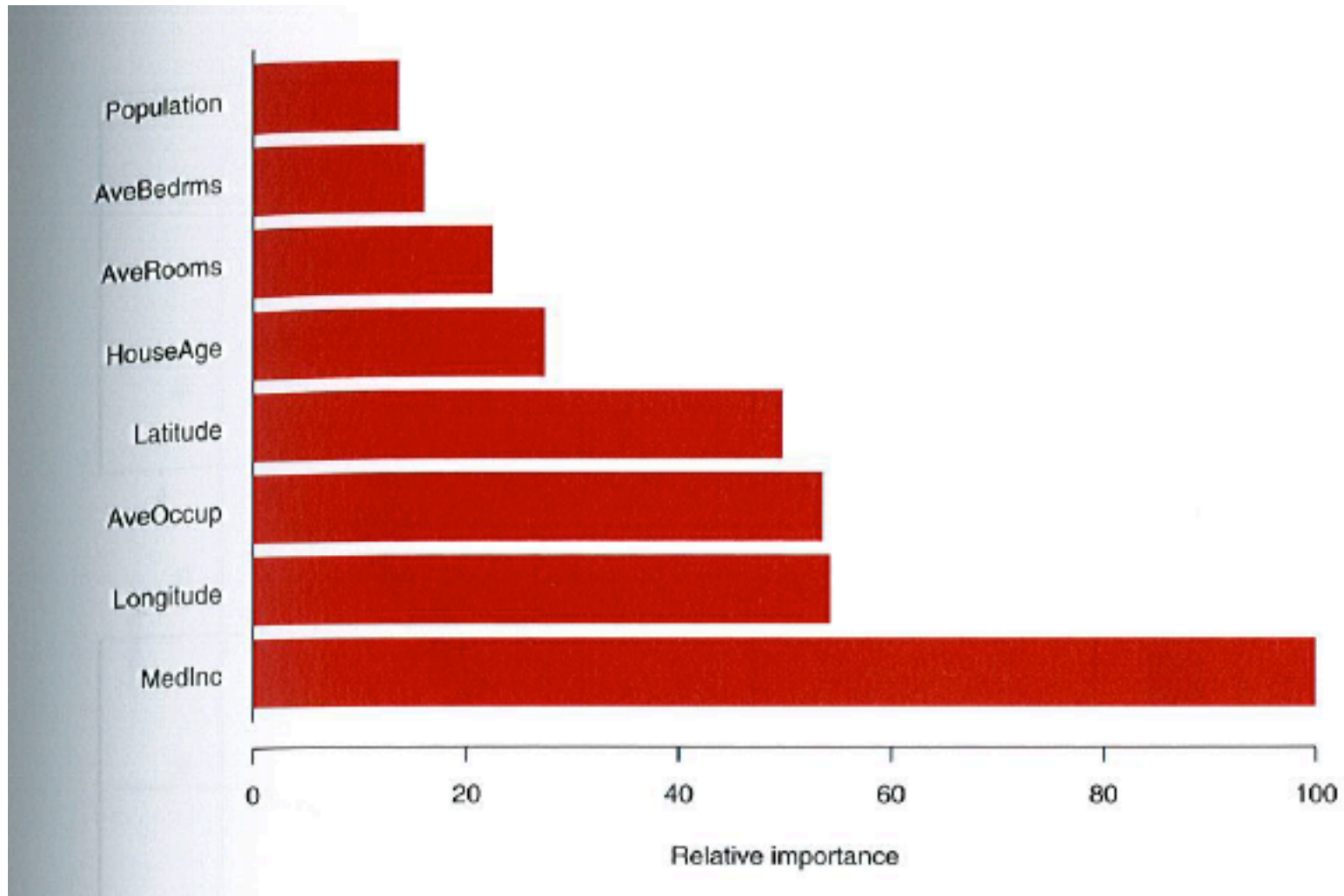
# Interpretation

- Interpretation is lost with boosting.
- We need to rely on alternative metrics to reveal the model structure.

## **Relative Importance of a Predictor variable.**

- Often many nuisance predictors which cloud model interpretation.
- Important to understand how important a variable is in predicting response, and dividing classes.

# Relative Importance



# Summary

- Boosting trees is complicated.
- Steepest descent a good idea... but prone to being too “greedy” by nature of the step.
- Gradient boosting a “less greedy” alternative.
- Main idea: fit trees to “residuals” defined as gradient function evaluations. Update functional approximation in an iterative manner. (Quasi-Newton like).
- Several improvements can be made to gradient boosting, including: subsampling, and shrinkage.
- Lots of parameters have to be set: number of trees, depth, subsampling, and shrinkage!

But..... It really works!