Statistical Machine Learning

Lecture 9: Boosting and Additive Trees

W.Q.Cui Research Group

Department of Statistics and Finance
University of Science and Technology of China

2018 Autumn

Contents

Boosting Methods

- One of the most powerful learning ideas introduced in the last 10 years
- Classification and Regression
- Combines "weak" classifiers to produce a powerful "committee"
  - Resemblance to bagging and committee-based approach
- First introduced by Freund and Schapire in 1996 (ICML)

Premises

- Two Class Problem
- Vector of Predictor Variables X
- $G(X)$ takes one of the two values $\{-1, 1\}$
- Error Rate:

$$\widetilde{err} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i))$$

Premises

- Expected Error Rate on Future Predictions:

$$E_{XY} I(Y \neq G(X))$$

- Weak Classifiers – performs slightly better than random guessing

AdaBoost.M1

- Introduced by Freund and Schapire (1997)
- **Approach**:
  - Sequentially apply the weak classifier to repeatedly modified versions of the data
- **Result**:
  - Sequence of Weak Classifiers (M total)

$$G_m(x), m = 1, 2, \cdots, M$$

  - Vastly improved classification performance

AdaBoost.M1

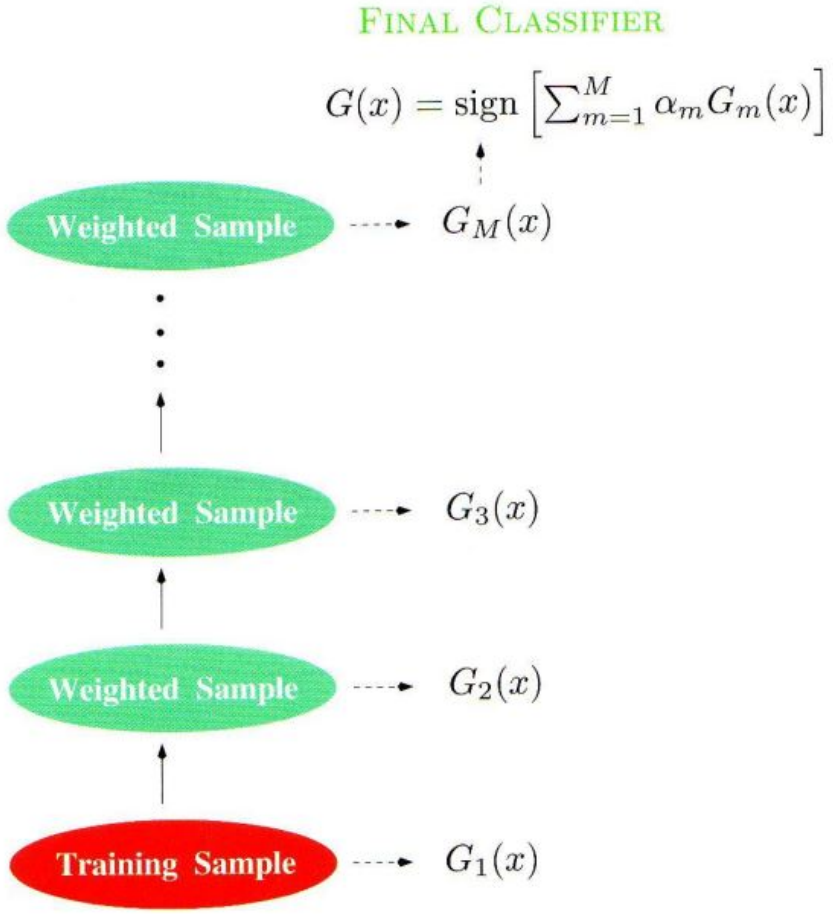


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

AdaBoost.M1

- $\alpha_1, \alpha_2, \cdots, \alpha_M$ weighs contribution of each classifier
- Data modification at each boosting step
- **Idea**:
  - Misclassified observations have their weights increased whereas those that are correctly classified have their weights decreased
- **Result**:
  - Difficult observations receive ever-increasing influence

Algorithm 10.1

Algorithm 10.1 *AdaBoost.M1.*

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

Notes

- AdaBoost.M1 known as "Discrete AdaBoost"
- Can be modified for real-value predictions ("Real AdaBoost")
- "Best off-the-shelf classifier in the world" (Breiman, 1998)

Boosting Fits an Additive Model

- Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions– "weak" classifiers

Basic Function Expansions

- Can be mathematically expressed as:

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

- β: expansion coefficients
- b: simple functions of multivariate argument x
- γ: parameters of b

Basic Expansion Example

- Single Hidden Layer Neural Network

$$b(x;\gamma) = \sigma(\gamma_0 + \gamma_1^t x)$$

- Wavelets (γ: location and scale shifts of "mother" wavelet)
- MARS (γ: variables and values for knots)
- Trees (γ: split variables, split points and predictions)

How to Solve these Functions?

- Typically, we minimize a loss function L
  - Squared-error
  - Likelihood-based loss function

$$\min_{\{\beta_m,\gamma_m\}_{m=1}^{M}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

- Requires computationally intensive numerical optimization techniques

Forward Stagewise Additive Modeling

- Solves the subproblem by fitting just one single basis function at a time

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

- Sequentially add new basis functions without changing those already added

Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward stagewise additive modeling.*

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 $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Loss Functions

- One could use "least squares", but for classification's sake, it's not a very good choice (when too good becomes bad)
- AdaBoost.M1 is equivalent to FSAM with an exponential loss function

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

- Proof in pages 305-6 that AdaBoost.M1 indeed minimizes the exponent loss

AdaBoost.M1 and Exponential Loss

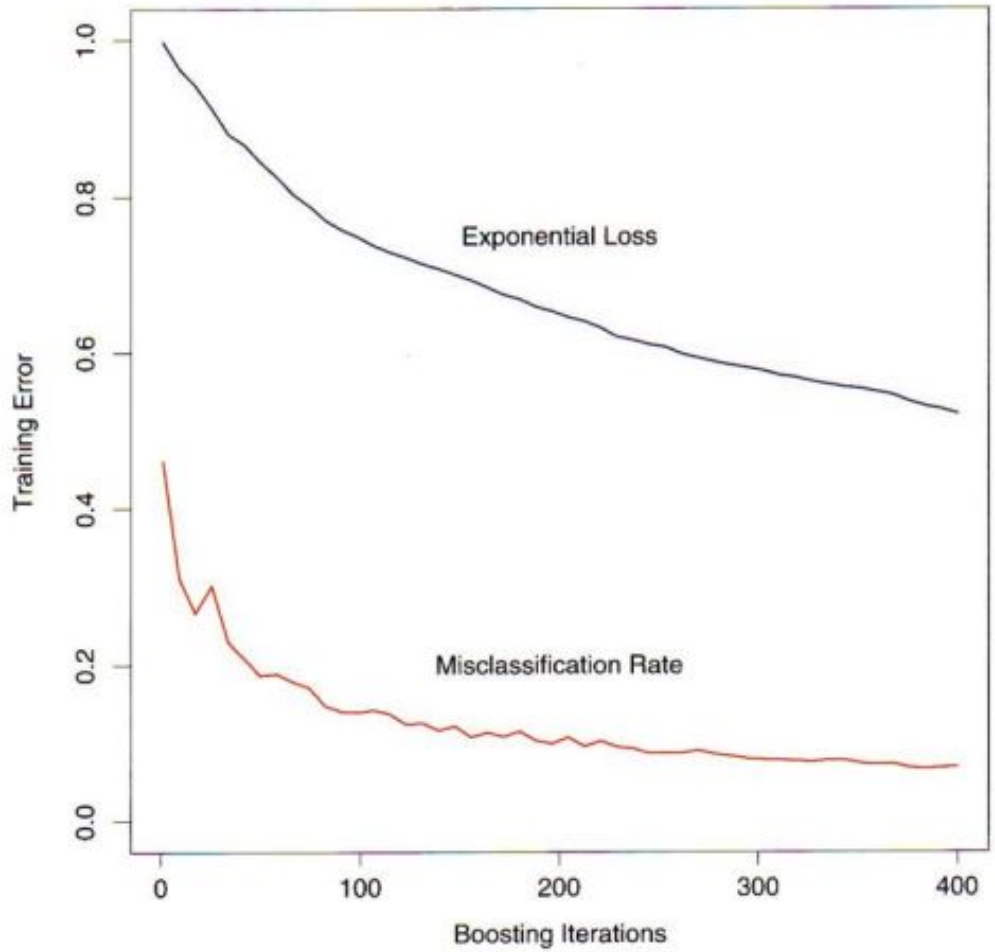


FIGURE 10.3. *Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss:* $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$.

Why Exponential Loss?

- **So**
  - AdaBoost.M1 = FSAM with Exponential Loss
- **Next**
  - Could we use something else with FSAM?

What does Exponential Loss Estimate?

- Friedman et al., 2000 showed that AdaBoost is estimating one-half the log-odds of $Pr(Y = 1|x)$

$$f^*(x) = \arg\min_{f(x)} E_{Y|x}(e^{-Yf(x)})$$
$$= \frac{1}{2} log \frac{Pr(Y = 1|x)}{Pr(Y = -1|x)}$$
$$Pr(Y = 1|x) = \frac{1}{1 + e^{-2f^*(x)}}$$

- This is answered by the population minimizer property of the loss function

Another Loss Criterion

- Binomial Negative Log-likelihood or deviance (or cross-entropy)
- **Definitions**:
  - Take $p(x)$ to be $\frac{1}{e^{-2f^*(x)}}$
  - $Y' = (Y+1)/2 \in \{0,1\}$
- **Loss Function**:

$$log(1 + e^{-2Yf(x)})$$

Loss Functions and Robustness

Loss Functions for Classification

- Misclassification
- Exponential
- Binomial Deviance
- Squared Error
- Support Vector

Robust Loss Functions for Classification

- **Margin**: $yf(x)$
- **Positive margin**: correctly classified
- **Negative margin**: incorrectly classified
- **Goal of Classification**: produce positive margins as frequently as possible
- **Loss Function**: Penalize negative margins and not positive ones (squared error!!!)
  - Monotonous decreasing functions

Robust Loss Functions for Classification

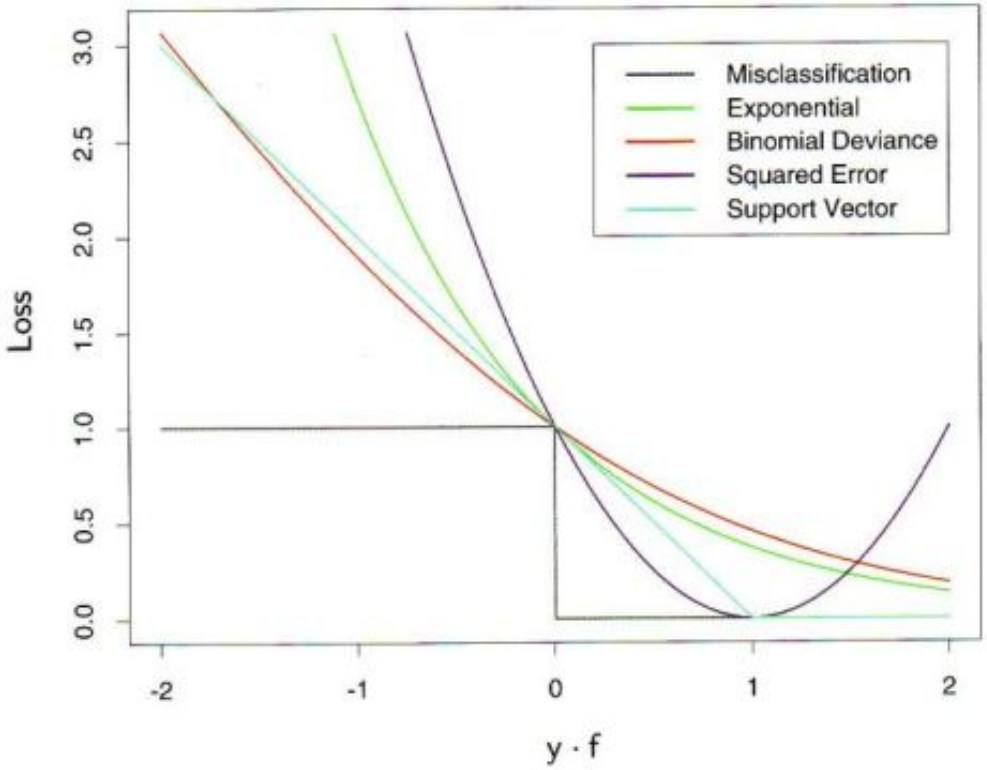


FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1+\exp(-2yf))$*; squared error:*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

Analysis

- Exponential and Deviance Loss are monotone continuous approximations to misclassification loss
- $\Uparrow$ reward for positive margins
- $\Downarrow$ penalize negative ones
- Binomial deviance increases **linearly** for large negative margins
- Exponential criterion increases **exponentially**

Analysis

- Exponential criterion is thus sensitive to nois data (e.g. misspecification of class labels in training data)
- Squared-error loss is not a good surrogate for misclassification error since it places increasing influence are correctly classified observations

What about Loss Functions for K classes?

- Generalization of K-class multinomial deviance loss function:

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = G_k) f_k(x) + log\left(\sum_{\ell=1}^{K} e^{f_\ell(x)}\right)$$

- No known natural generalization of the exponential criterion for K classes

Robust Loss Functions for Regression

- **Squared Error (L2)**
  - Performance severely degrades for long-tailed error distributions, such as "outliers"
- **Absolute Error (L1)**
  - More robust but less efficient for Gaussian errors
- **Huber**
  - Proposed in 1964

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2, & \text{if } |y - f(x)| \leq \delta \\ \delta(|y - f(x)| - \delta/2), & \text{otherwise.} \end{cases}$$

Comparison of Regression Loss Functions

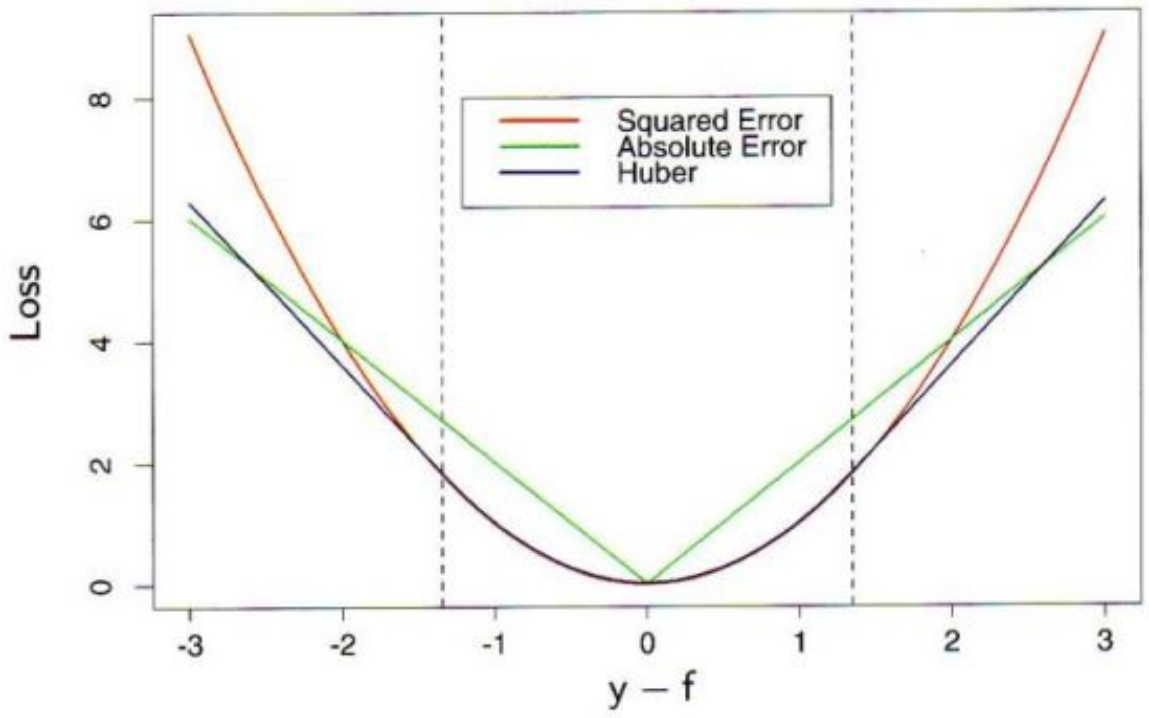


FIGURE 10.5. *A comparison of three loss functions for regression, plotted as a function of the margin $y-f$. The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when $|y-f|$ is large.*

Analysis

- Squared-error loss for regression and Exponential loss for classification are not robust from a statistical standpoint
- However, they give simple, elegant modular boosting algorithms in the context of FSAM
- Huber and Binomial deviance, however robust, does not have this property
- Section 10.10.2: how to derive simple, elegant boosting algorithms based on any differentiable loss criterion

"Off-the-Shelf" Procedures for Data Mining

- Challenge
  - Many methods
  - Even more situations
  - Not enough time
- What to choose? (NN, SVM, Trees, MARS, kNN, kernels, etc.)

Requirements for "Off-the-Shelf" Methods

- **Speed**
  - Can crunch lots of observations with lots of variables
  - Computational Complexity of Model
- **Robustness**
  - Can handle messy data (continuous, discrete, cyclical, discretized)
  - Missing values
  - Outliers
  - Scaling

Requirements for "Off-the-Shelf" Methods

- Interpretability
  - Not enough to simply produce predictions
  - Want qualitative understanding of relationship between joint values and resulting predicted response value
  - Black box methods such as NN are far less useful (unless for purely predictive settings such as pattern recognition)

Comparsions on Major Learning Methods

TABLE 10.1. *Some characteristics of different learning methods. Key:* ● *= good,* ● *=fair, and* ● *=poor.*

Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels
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	●	●	●	●	●

Off-the-Shelf Method: Winner

- Decision Tree Learning

- Relatively fast to construct;
  interpretable models;
  naturally incorporate mixtures of numeric and categorical predictor variables;
  handle missing values elegantly;
  invariant under (strictly monotone) transformation of the individual predictors;
  immune to outliers;
  internal feature selection as part of training

- Tress have emerged as the most popular learning method because of these reasons

The Problem

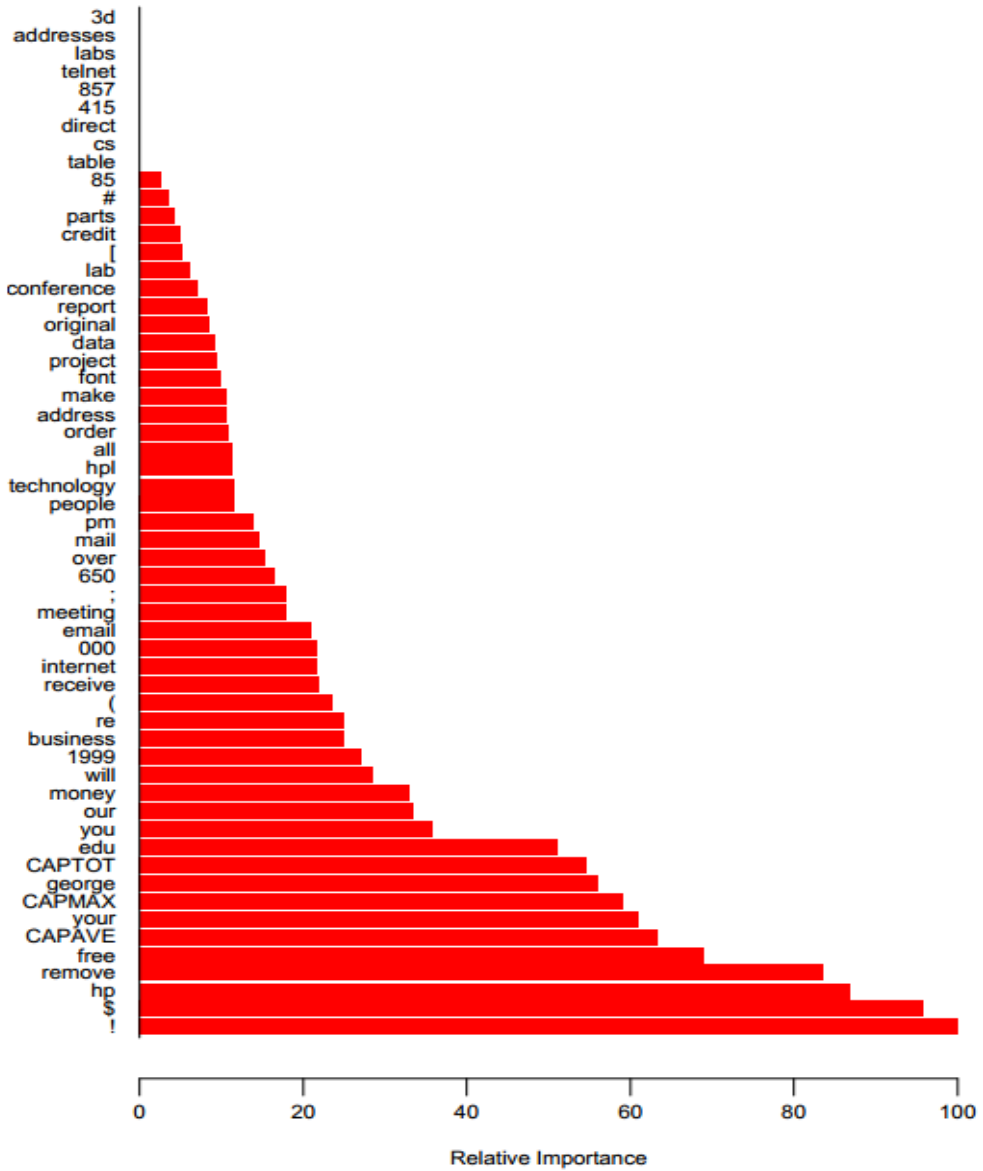
- Inaccuracy
- Seldom achieve the best that is possible with the data at hand
- Boosting comes to the rescue
  - Improves accuracy
  - Maintains interpretability (somewhat)
  - Requires M times longer to train
  - Loses robustness to mislabeled data (AdaBoost specifically)
- Multiple Additive Regression Tree (MART) tries to mitigate these problems

Example — Spam Data

- Let's see MART in action first before going into details
- Spam dataset from Chapter 9
- **Error Rates**:
  - MART: 4.0%
  - Additive Logistic Regression: 5.3%
  - CART (fully grown and pruned by CV): 8.7%
  - MARS: 5.5%

(standard error of estimates: 0.6%)

Relative Importance Measure of Predictors



Relative Importance Measure

- More on this in Section 10.13
- 57 Predictor Variables
- Most Relevant:
  - !
  - $
  - hp
  - remove
- Least relevant:
  - 857
  - 415
  - table
  - 3d

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

Partial Dependence

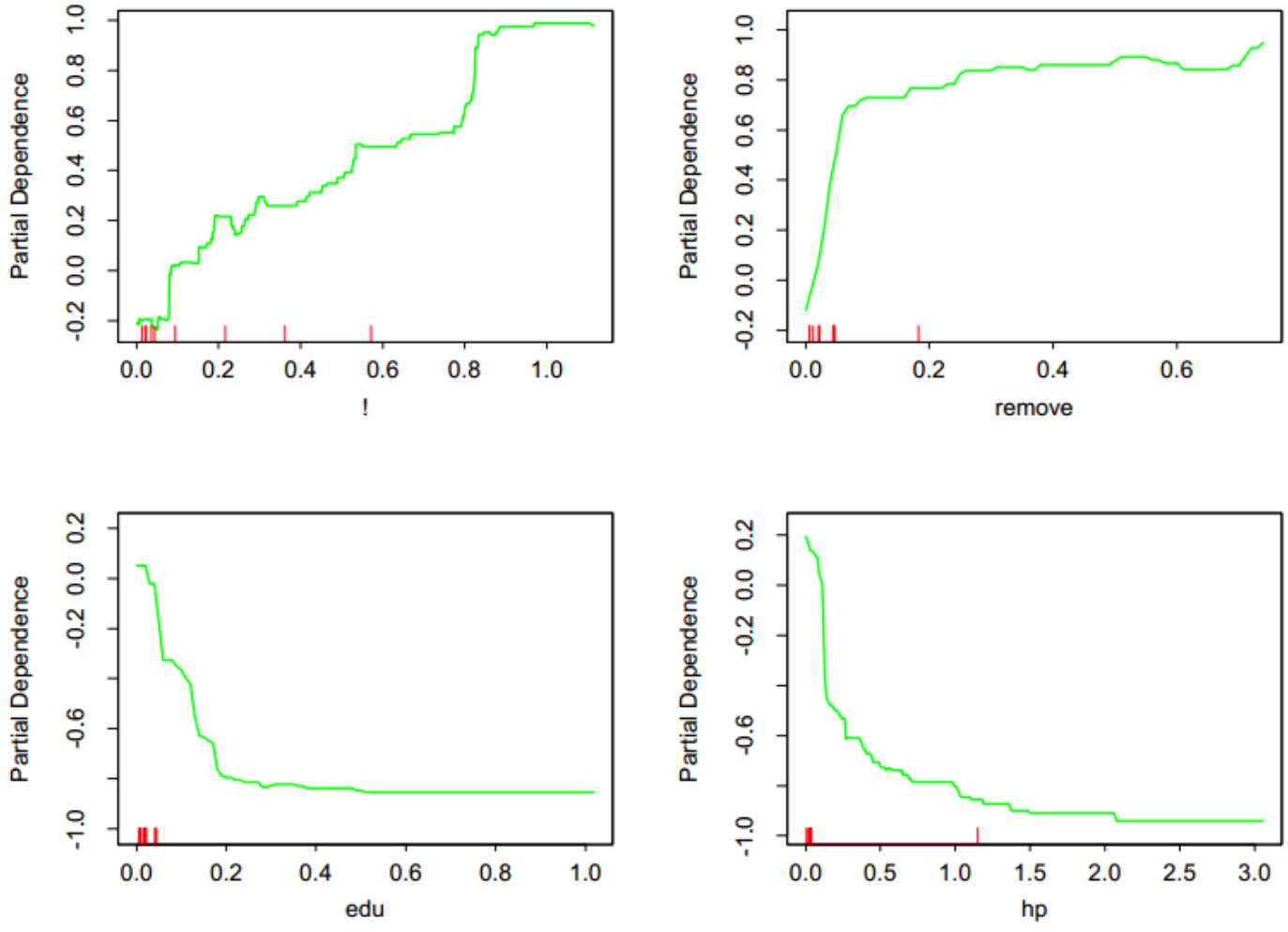


FIGURE 10.7. *Partial dependence of log-odds of* `spam` *on four important predictors. The red ticks at the base of the plots are deciles of the input variable.*

Partial Dependence

- One Variable
  - Shows dependence of log-odds with predictor
- Two Variable
  - Shows interactions among the predictor variables
  - When to Run?
    - Running MART with $J = 2$(main effects model) yields a higher error rate when compared to running with larger J

Partial Dependence

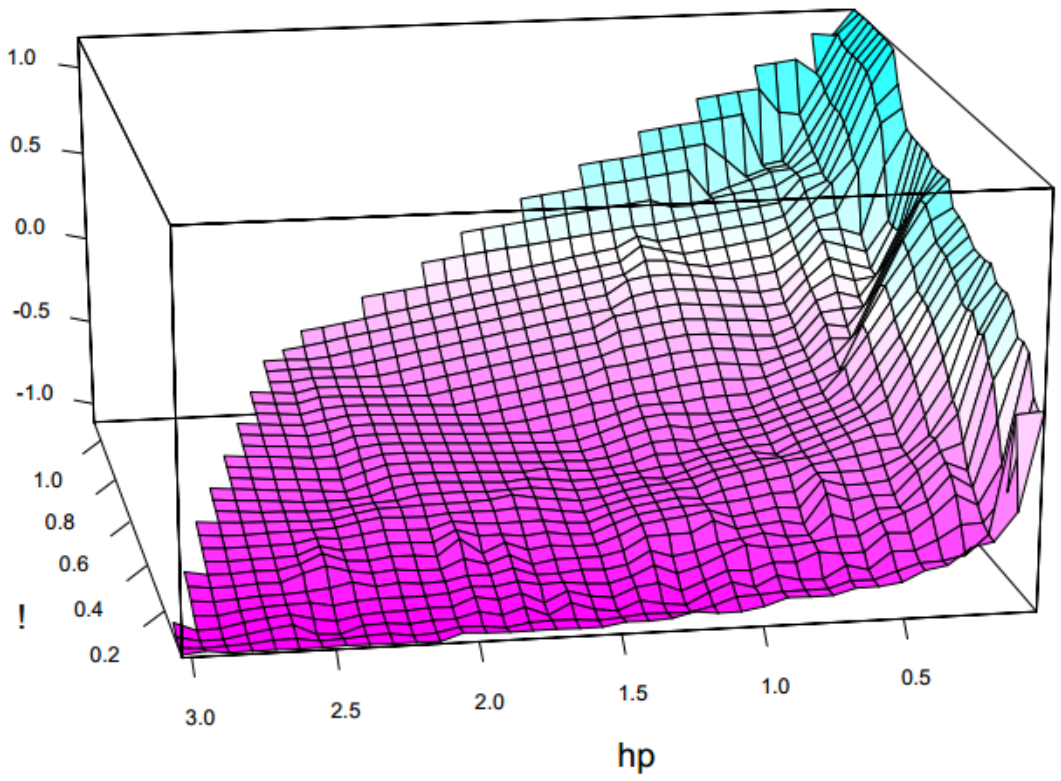


FIGURE 10.8. *Partial dependence of the log-odds of* `spam` *vs.* `email` *as a function of joint frequencies of* `hp` *and the character* `!`.

Boosting Trees

- Decision tree : $x \in R_j \Longrightarrow f(x) = \gamma_j$
- Formal Expression:

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

- Parameter : $\Theta = \{R_j, \gamma_j\}_1^J$
- Optimization Process:

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

Boosting Trees

- Approximation
  * Finding γ_j given R_j
    - Trivial
    - Estimating γ_j is often the mean/mode of y in region R_j
  * Finding R_j
    - Difficult
    - Typical way is to use a greedy, top-down recursive partitioning algorithm
    - Can also approximate by a smoother and more convenient criterion

Boosted Trees

- Sum of Trees

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

- Solve using FSAM

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

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

- The difficult part is finding R_{jm}

Solving the FSAM Problem

- Some special cases are easier
  - Square-error loss: find the tree that best predict the current residual
  - Two-class w/ Exponential loss: $Adaboost.M1$; tree that minimize weighted error rate;$\{-1, +1\}$
  - N-class w/ Exponential loss:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} \omega_i^{(m)} \exp[-y_i T(x_i; \Theta_m)]$$

    * γ can be found by (10.31)–weighted log-odds in each region

Solving the FSAM Problem

- Regression: Absolute Error, Huber Loss
- Classification: Deviance
- Will robustify boosting trees
- However, they do not give rise to simple fast boosting algorithms

Numerical Optimization

- Solving each "step" in FSAM by numerical optimization
- Differentiable loss criterion
- Total loss:

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

- Goal:

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

Numerical Optimization

- f is a vector
- "Parameters" of f are the values at each data point

$$f = \{f(x_1), f(x_2), \cdots, f(x_N)\}$$

- Numerical optimization solves the problem with a sum of component vectors

$$f_M = \sum_{m=0}^{M} h_m \qquad f_0 = h_0$$

Steepest Descent

- Greedy Strategy

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

$$\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)$$

$$f_m = f_{m-1} - \rho_m g_m$$

- Gradients in Table 10.2

Gradient Boosting

- Simplifying

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

- To

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

- Rationale: Minimize Loss vs. Generalization

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 $i = 1, 2, \ldots, N$ 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 $\hat{f}(x) = f_M(x)$.

Right-Sized Trees for Boosting

- Size of tree (J: number of terminal nodes) for each iteration of boosting
- Simple strategy: constant J
- How to find J?
  - Minimize prediction risk on future data

ANOVA

- Analysis of Variance of Predictor Variables

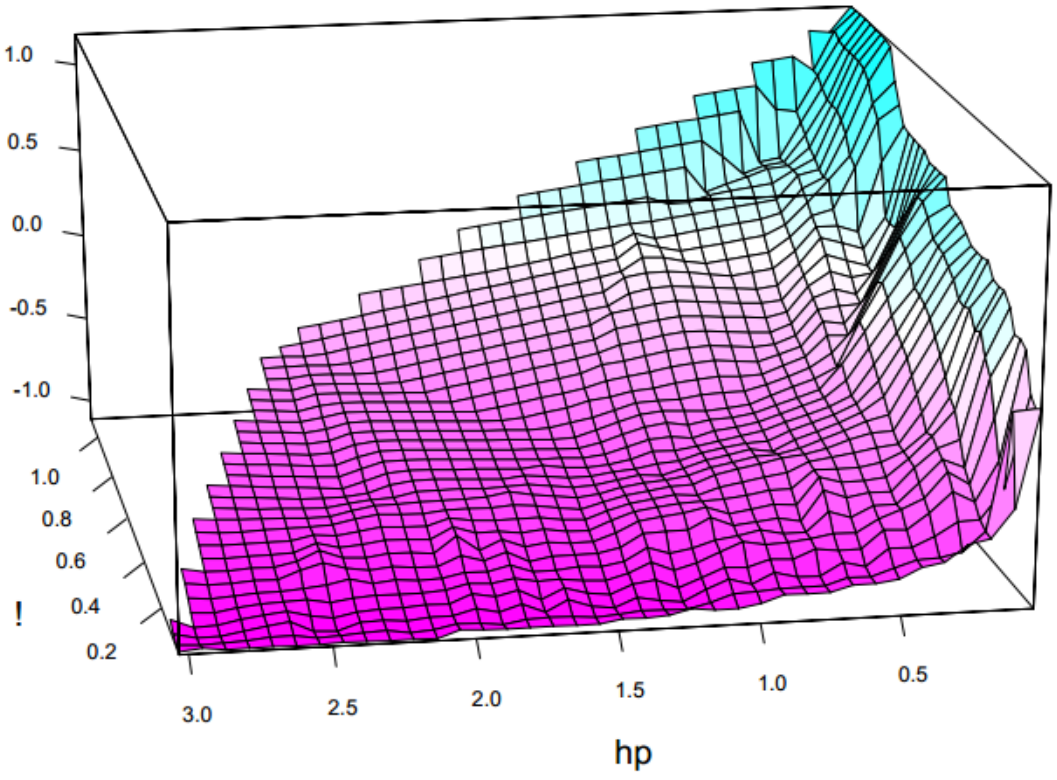


FIGURE 10.8. *Partial dependence of the log-odds of* `spam` *vs.* `email` *as a function of joint frequencies of* `hp` *and the character* `!`.

Tree Size

- Most problems have low-order interaction effects dominating the problem space
- Thus, models with high-order interaction will suffer in accuracy
- Interaction effects are limited by J
  - No interaction effects of level greater than $K-1$ are possible
  - $J=2$: Decision Stump (only main effects, no interactions)
  - $J=3$: two-variable interaction effects are allowed

Tree Size Comparison

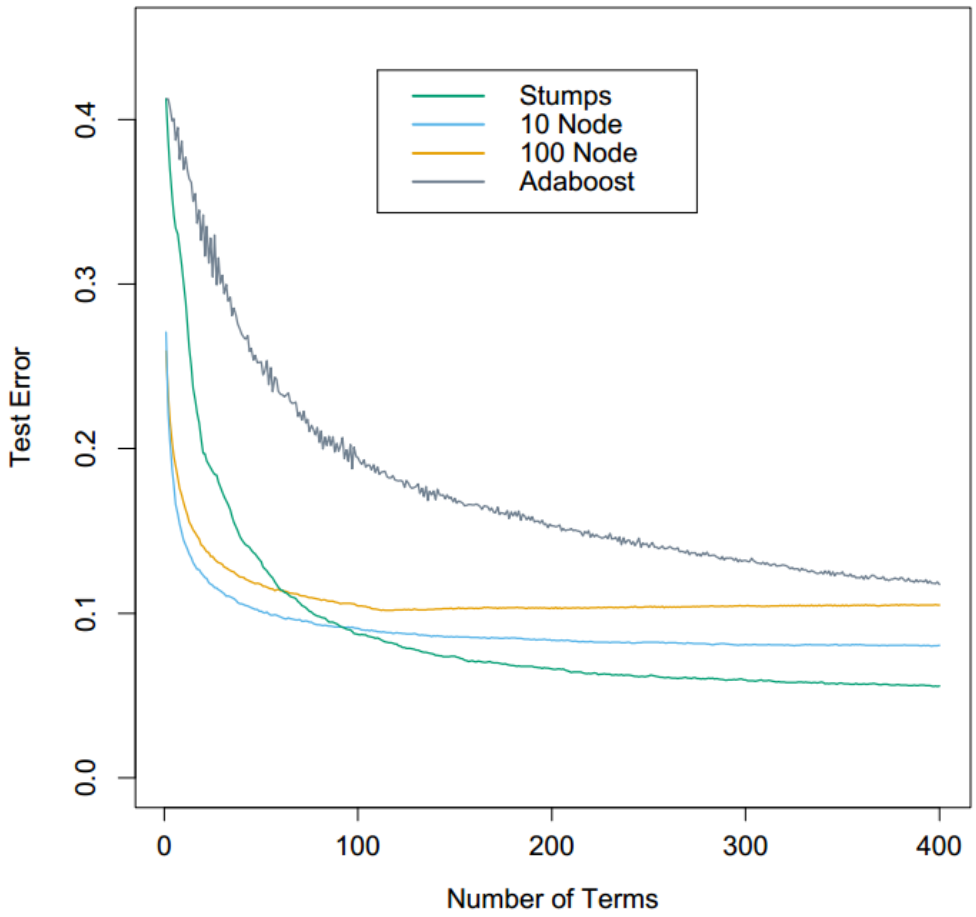


FIGURE 10.9. *Boosting with different sized trees, applied to the example (10.2) used in Figure 10.2. Since the generative model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost Algorithm 10.1.*

Choosing a J

- Typically
  - $J = 2$ will be insufficient
  - $J > 10$ will be highly unlikely
  - $4 <= J <= 8$ works well in boosting by experience
  - $J = 6$ should be the initial guess

Regularization

- **Regularization**: prevention of overfitting of data by models
- Example: Parameter M
  - Increases M reduces the training risk
  - Could lead to overfitting
  - Use a hold-out set
    - Similar to early stopping strategy in NN

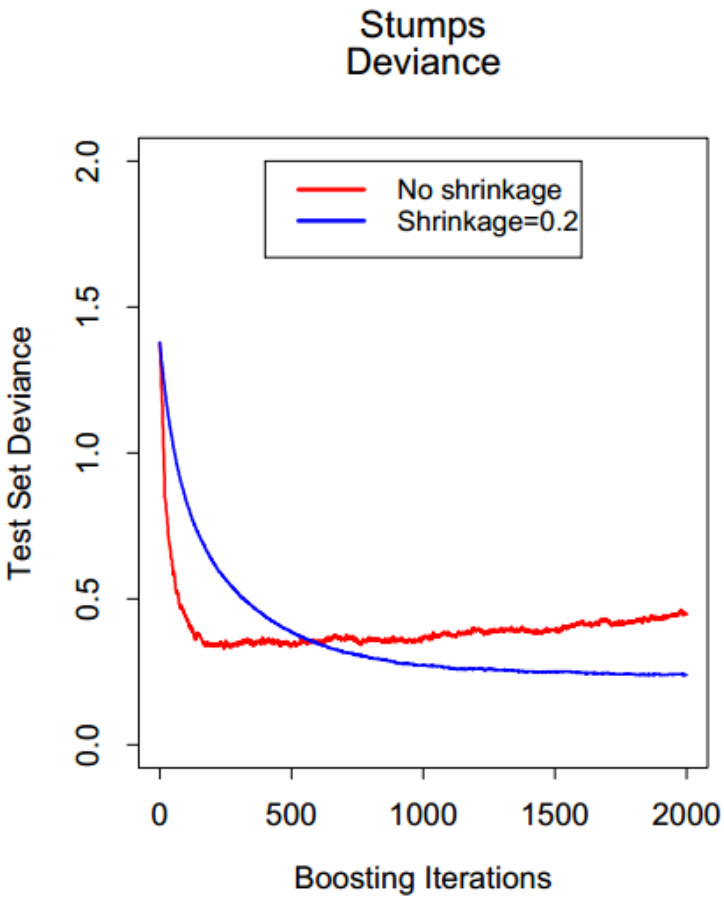
Shrinkage

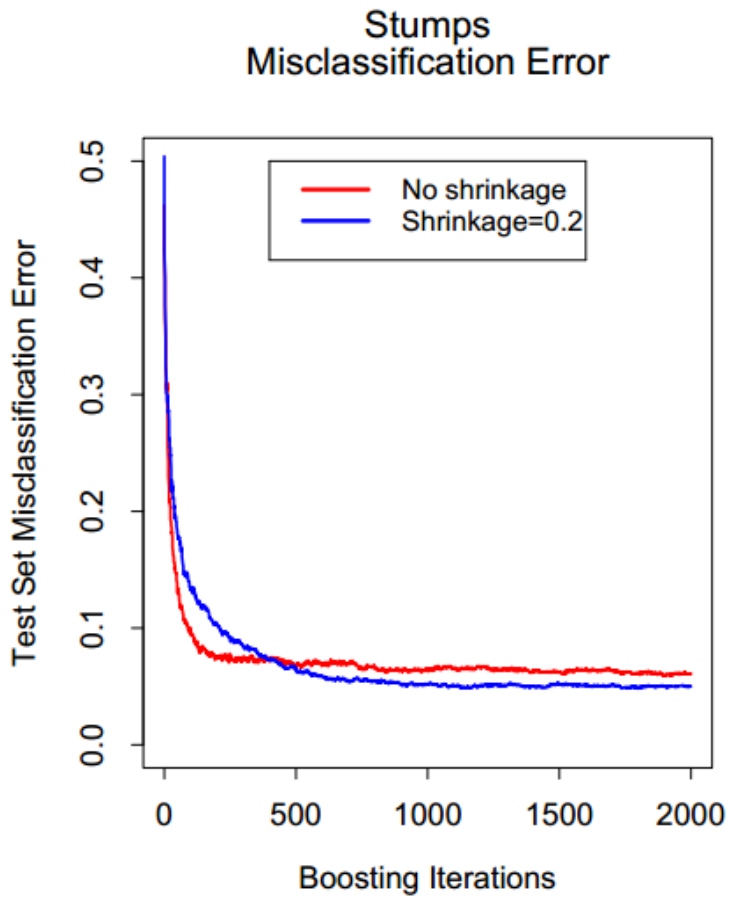
- Scale the contribution of each tree by a factor $0 < v < 1$

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

- Controlling the learning rate of the boosting procedure
- Empirically, smaller v favor better test error but longer training time
- Best strategy is to choose a small $v(v < 0.1)$ and find M by early stopping

Shrinkage vs. No Shrinkage





Shrinkage vs. No Shrinkage

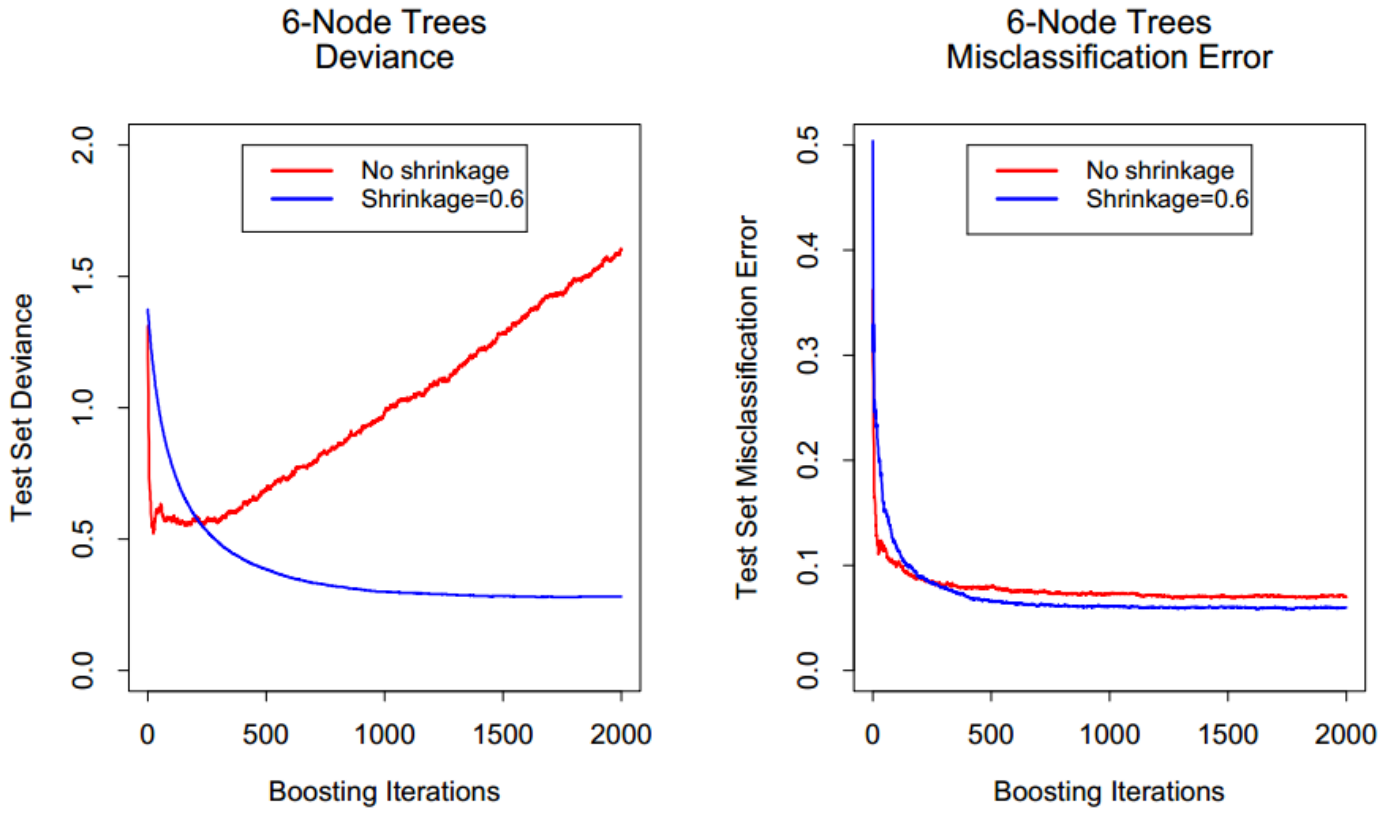


FIGURE 10.11. *Test error curves for simulated example (10.2) of Figure 10.9, using gradient boosting (MART). The models were trained using binomial deviance, either stumps or six terminal-node trees, and with or without shrinkage. The left panels report test deviance, while the right panels show misclassification error. The beneficial effect of shrinkage can be seen in all cases, especially for deviance in the left panels.*

Penalized Regression

- Consider the set of all possible J-terminal node regression trees as basis functions
- Thus, the linear model:

$$f(x) = \sum_{k=1}^{K} \alpha_k T_k(x)$$

- $K = cart(T)$ and is likely to be much larger than any possible training set
- Thus, penalized least squares is required to find the alphas

Penalized Regression

- Penalty Function
  - Ridge regression
  - Lasso

$$\hat{\alpha}(\lambda) = \arg\min_{\alpha} \left\{ \sum_{i=1}^{N} \left(y_i - \sum_{k} \alpha_k T_k(x_i) \right)^2 + \lambda \cdot J(\alpha) \right\}$$

$$J(\alpha) \sum_{k=1}^{K} \alpha_k^2$$

$$J(\alpha) \sum_{k=1}^{K} |\alpha_k|$$

Penalized Regression

- Many alphas will be zero with a large lambda
  - Only a fraction of possible tress are relevant
- Problem:
  - Still can't solve for all possible tress
- Solution:
  - Forward stagewise strategy
  - Initialize to $alpha = 0$ first
  - More iterations lead to smaller alphas

Penalized Regression

Algorithm 10.4 *Forward stagewise linear regression.*

1. Initialize $\hat{\alpha}_k = 0,\ k = 1, \ldots, K$. Set $\varepsilon > 0$ to some small constant, and M large.
2. For $m = 1$ to M:
   (a) $(\beta^*, k^*) = \arg\min_{\beta,k} \sum_{i=1}^{N} \left(y_i - \sum_{l=1}^{K} \alpha_l T_l(x_i) - \beta T_k(x_i) \right)^2$.
   (b) $\alpha_{k^*} \leftarrow \alpha_{k^*} + \varepsilon \cdot \text{sign}(\beta^*)$.
3. Output $f(x) = \sum_{k=1}^{K} \alpha_k T_k(x)$.

Penalized Regression in Action

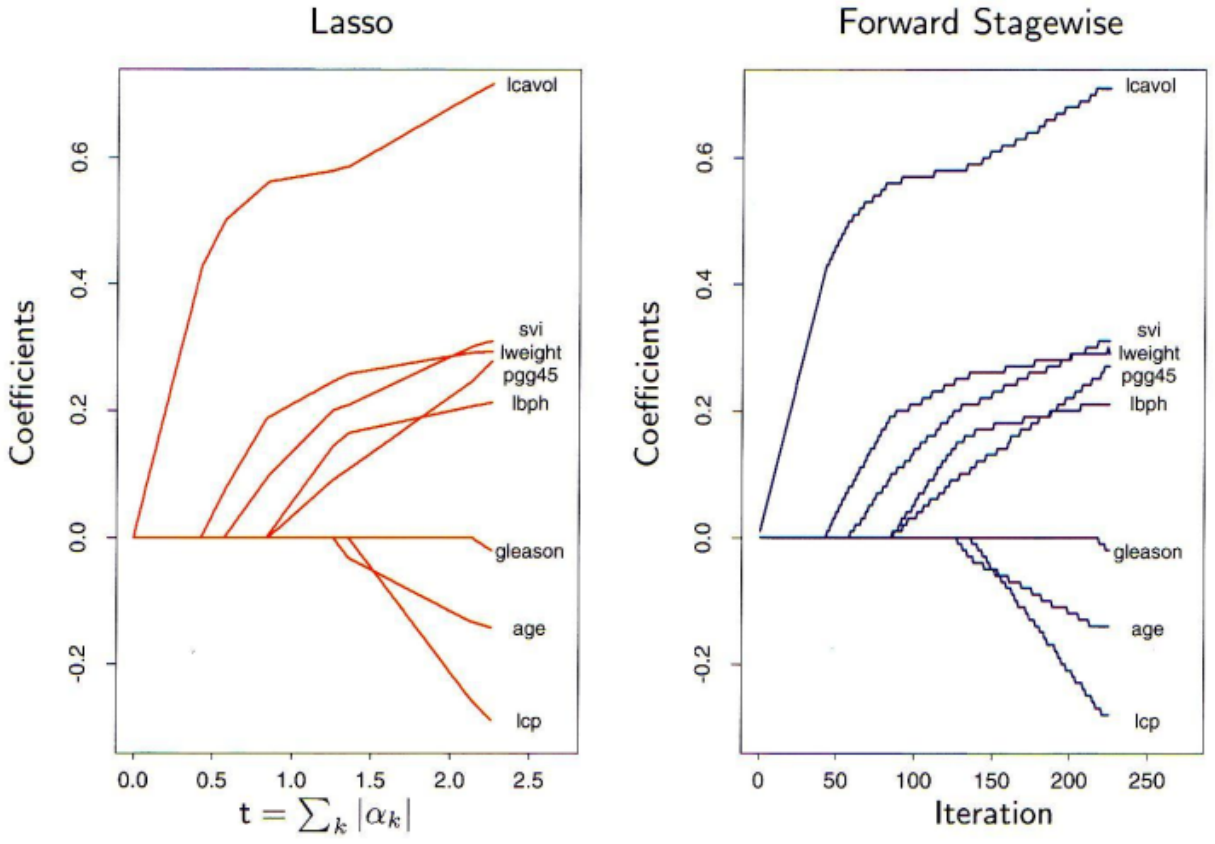


FIGURE 10.12. *Profiles of estimated coefficients from linear regression, for the prostate data studied in Chapter 3. The left panel shows the results from the lasso, for different values of the bound parameter* $t = \sum_k |\alpha_k|$. *The right panel shows the results of the stagewise linear regression algorithm 10.4, using* $M = 250$ *consecutive steps of size* $\varepsilon = .01$.

Analysis

- The approximation works (approximates lasso)
- Tree boosting with shrinkage resembles penalized regression
- No shrinkage is analogous to subset selection
  (penalizes the number of non-zero coefficients)

Virtues of the L_1 Penalty(Lasso) over L_2

- Superior performance of boosting over procedures such as SVM may be largely due to the implicit use of $L1$ versus $L2$ penalty
- $L1$ penalty is better suited to sparse situations(Donoho et al., 1995)
- Though minimization of $L1$-penalized problem is much more difficult than that for $L2$
- The forward stagewise approach provides an approximate, practical way to tackle the problem

Interpretation

- Single decision tress are highly interpretable
- Linear combination of tress lose this feature
- How to interpret the model then?

Relative Importance of Predictor Variables

- Breiman et al. (1984) proposed a measure of relevance for each predictor variable for a single decision tree
- Intuition: variable is the one that gives maximum estimated improvement in squared error risk
- Simply average over the trees for additive models
- Also works for K-class classifiers

Partial Dependence Plots

- Visualization is a great tool but is limited to low-dimensional views
- Marginal average of a model given a subset of input variables and the complement of that within all input variables
- Works for k-class problems as well