Boosting and Addative Modeling (part 3)

Matthew S. Shotwell, Ph.D.

Department of Biostatistics
Vanderbilt University School of Medicine
Nashville, TN, USA

April 1, 2020

Boosting trees

- denote a tress as follows

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

  where $\Theta = \{R_j, \gamma_j\}_1^J$ define the feature partition (i.e., the tree, R_j) and predictions within each region (γ_j)
- estimate Θ by minimizing average (or sum of) loss

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

- $\hat{\gamma}_j | \hat{R}_j$ is easy (often $\hat{\gamma}_j = \bar{y}_j$)
- $\hat{R}_j$ is hard (combinatorial; use recursive partitioning)

Boosting trees

- boosted tree model, using FSAM

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

- at each stage, must solve

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

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$

Boosting trees

- given regions R_{jm}, finding optimal γ_{jm} is usually easy

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

- again, finding regions is difficult
- for squared-error loss, no more difficult than unboosted tree
- for two-class problem and exponential loss $\rightarrow$ AdaBoost.M1
- for most other problems, no simplification possible

Gradient boosting

- need to solve this:

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

- if L is differentiable, can think about solving using numerical optimization
- rewrite the average loss as follows

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

- minimize $L(f)$ w.r.t. f, where $f(x)$ is constrained to be sum of trees

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

Gradient boosting

- ignoring the tree structure for the moment, can think about numerically optimizing

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

  where the "parameters" $f \in \mathbb{R}^N$ are the values of the approximating function $f(x_i)$ at each of the N training points x_i

$$f = \{f(x_1), \ldots, f(x_N)\}^T$$

- can find f_M sequentially by adding committee members h_m:

$$f_M = \sum_{m=0}^{M} h_m, \quad h_m \in \mathbb{R}^N$$

  where h_0 is an initial guess
- how to find h_m ?

Gradient boosting

- find h_m using steepest descent
- choose $h_m = -\rho_m r_m$ for scalar ρ_m and $r_m \in \mathbb{R}^N$ is gradient of $L(f)$ evaluated at $f = f_{m-1}$

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

- step length ρ_m is either fixed or optimized:

$$\rho_m = \operatorname{argmin}_\rho L(f_{m-1} - \rho r_m)$$

- the model/committee is updated as

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

- this is a greedy strategy

Gradient boosting

- back to boosted trees; at each step, find new committee tree:

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

- difficult to solve directly
- **gradient boosting finds an approximate solution by finding a tree $T(x; \Theta_m)$ that moves f_{m-1} in direction of negative gradient of L evaluated at f_{m-1}, denoted r_m**
- **fit a tree $T(x_i; \Theta_m)$ to the negative gradient values:**

$$\tilde{\Theta}_m = \text{argmin}_{\Theta} \sum_{i=1}^{N} (-r_{im} - T(x_i; \Theta))^2$$

- note: squared error loss function
- $\tilde{\Theta}_m$ not identical to $\hat{\Theta}_m$ close enough!

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

Spam Data

- 4601 email messages
- all classified as ‘email’ or ‘spam’ (junk email)
- 57 features; relative frequencies of common words and punctuation
- problem: classify new emails as ‘email’ or ‘spam’
- classification loss probably not symmetric (why?)

TABLE 1.1. *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`	`!`	`our`	`re`	`edu`	`remove`
`spam`	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
`email`	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Tree size affects performance

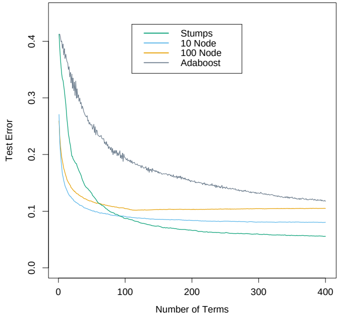


FIGURE 10.9. *Boosting with different sized trees, applied to the example (10.2) used in Figure 10.2. Since the generative model is additive, stumps perform the best. The boosting algorithm used the binomial deviance loss in Algorithm 10.3; shown for comparison is the AdaBoost Algorithm 10.1.*

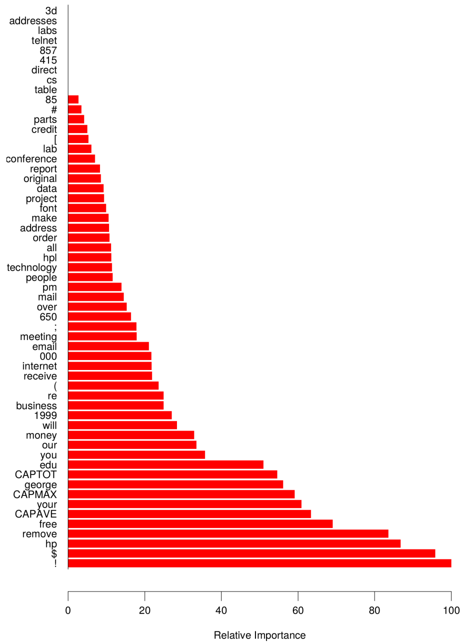
Gradient boosting implementation

- R package `gbm` (for “gradient boosted models”)

Gradient boosting of SPAM data

method	test error
gradient boosting	4.5%
add. logis. reg.	5.5%
CART	8.7%
MARS	5.5%

Relative importance of 57 spam features



Partial dependence

- let x_s be a subset of predictors and x_c it's complement
- usually x_s has just one or two dimensions, so we can plot them
- a predictor f depends on both $f(x_s, x_c)$
- partial dependence is the prediction at a value of x_s, averaged over the values that x_c could take:

$$f_s(x_s) = \frac{1}{N} \sum_{i=1}^{N} f(x_s, x_{ic})$$

where x_{ic} are from the trainind data

Partial dependence of log-odds of spam

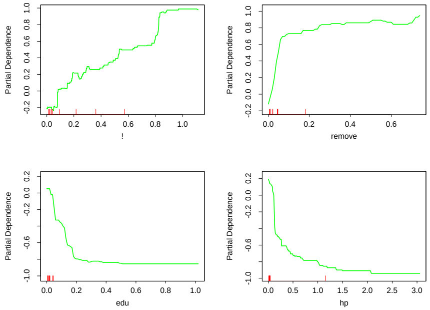


FIGURE 10.7. *Partial dependence of log-odds of* `spam` *on four important predictors. The red ticks at the base of the plots are deciles of the input variable.*

Partial dependence of log-odds of spam

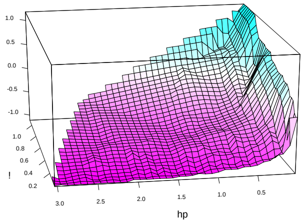


FIGURE 10.8. *Partial dependence of the log-odds of* `spam` *vs.* `email` *as a function of joint frequencies of* `hp` *and the character* `!`*.*