Boosting and Additive Trees

Thomas Lonon

Division of Financial Engineering
Stevens Institute of Technology

April 3, 2017

Consider a two-class problem with output variable $Y \in \{-1, 1\}$. Given a vector of predictor variables X and a classifier $G(X)$ taking one of the two values.

The error rate on the training sample is given by:

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

A weak classifier is one whose error rate is only slightly better than random guessing.

Purpose of boosting is to sequentially apply the weak classification algorithm to modified version of the data.

This produces a sequence of weak classifiers $G_m(x), m = 1, \ldots, M$. These are then combined using a weighted majority to produce the final prediction:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

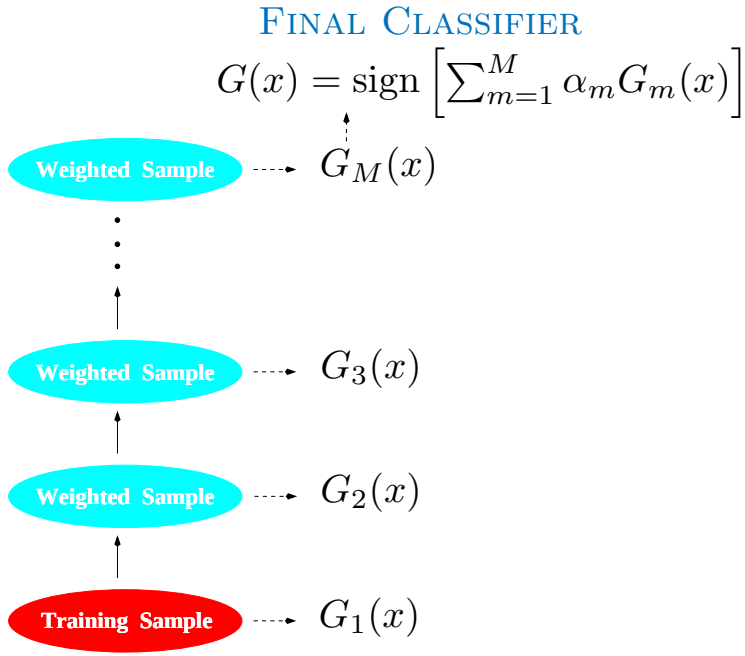

FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Algorithm 10.1: AdaBoost.M1

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1, \ldots, N$
2. For $m = 1$ to M :
   - 2.1 Fit a classifier $G_m(x)$ to the training data using weights w_i
   - 2.2 Compute:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}_{\{y_i \neq G_m(x_i)\}}}{\sum_{i=1}^{N} w_i}$$

   - 2.3 Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
   - 2.4 Set $w_i \leftarrow w_i e^{\alpha_m \mathbb{I}_{\{y_i \neq G_m(x_i)\}}}, i = 1, \ldots, N$
3. $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

[2]

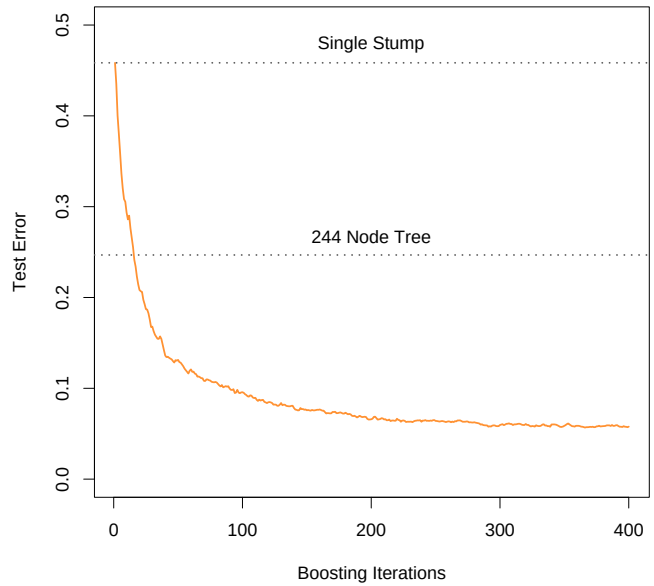

FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a* 244*-node classification tree.*

Let the individual classifiers, $G_m(x) \in \{-1, 1\}$ form the basis functions. More generally, these take the form:

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

where $\beta_m, m = 1, 2, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by parameters γ.[2]

- In single-hidden-layer neural networks, $b(x;\gamma) = \sigma(\gamma_0 + \gamma_1^T x)$, where $\sigma(t) = 1/(1+e^{-t})$ is the sigmoid function, and γ parameterizes a linear combination of the input variables
- Multivariate adaptive regression splines uses truncated-power spline basis functions where γ parameterizes the variables and values for the knots.
- For trees, γ parameterizes the split variables and split points at the internal nodes, and the predictions at the terminal nodes

[2]

Typically these models are fit by minimizing a loss function averaged over the training data:

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

Or a simple alternative being fitting a single basis function:

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

[2]

Algorithm 10.2: Forward Stagewise Additive Modeling

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M
   2.1 Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

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

[2]

For squared-error loss:

$$L(y, f(x)) = (y - f(x))^2$$

we have:

$$\begin{aligned} L(y_i; f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2 \end{aligned}$$

where $r_{im} = y_i - f_{m-1}(x_i)$ is the residual of the current model on the i^{th} observation

If we use exponential loss:

$$L(y, f(x)) = e^{-yf(x)}$$

then AdaBoost.M1 is equivalent to forward stagewise additive modeling.

We use the basis functions as the individual classifiers, and so using the exponential loss function, we have to solve:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(x_i)+\beta G(x_i))}$$

for the classifier G_m and the coefficient β_m

We can express this as:

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

with $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$. Notice that this doesn't depend on either β or $G(x)$.

This solution can be obtained in two steps. First for $\beta > 0$ we have:

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}_{\{y_i \neq G(x_i)\}}$$

Our criterion in (β_m, G_m) becomes:

$$e^{-\beta} \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

which can be written as:

$$(e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}_{\{y_i \neq G(x_i)\}} + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}$$

If we plug this G_m into the earlier equation, we have:

$$\beta_m = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

where

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}_{\{y_i \neq G_m(x)\}}}{\sum_{i=1}^{N} w_i^{(m)}}$$

The approximation is then updated

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

Why Exponential Loss?

For exponential loss, the primary attraction is computational. It leads to the simplest AdaBoost algorithm.

The population minimizer for exponential loss is:

$$f^*(x) = \arg\min_{f(x)} \mathbb{E}_{Y|x}[e^{-Yf(x)}] = \frac{1}{2}\log\frac{\mathbb{P}(Y=1|x)}{\mathbb{P}(Y=-1|x)}$$

or

$$\mathbb{P}(Y=1|x) = \frac{1}{1+e^{-2f^*(x)}}$$

Robust Loss Functions for Classification

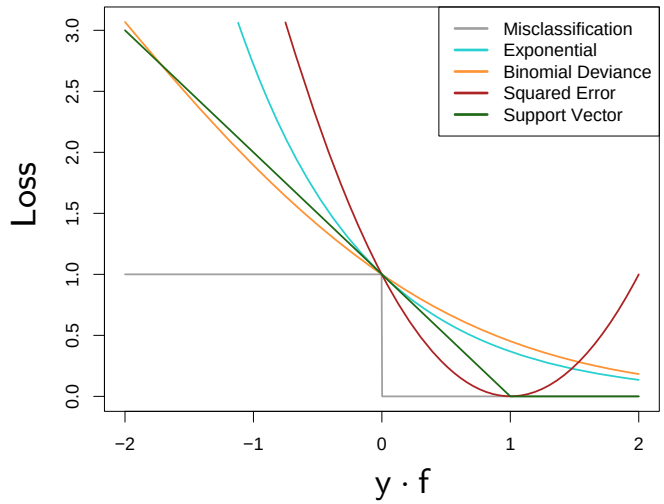

FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:*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

Robust Loss Functions for Regression

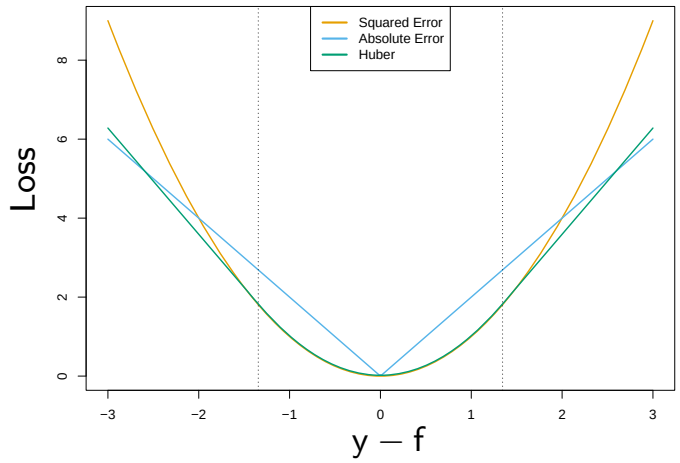

FIGURE 10.5. *A comparison of three loss functions for regression, plotted as a function of the margin $y-f$. The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when $|y-f|$ is large.*

As a reminder, once we have partitioned a space into regions $R_j, j = 1, \ldots, J$, we then assign a constant γ_j to each region such that our predictive rule is:

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

So our tree can be formally expressed as:

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

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

Suboptimal Problems

To find these parameters, we minimize the empirical risk:

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

Finding γ_j given R_j: Given the R_j, estimating γ_j is typically trivial and often $\hat{\gamma} = \bar{y}_j$

Finding R_j: The difficult part. Typical strategy is to use a greedy, top-down recursive algorithm to find the R_j

The boosted tree model is a sum of such trees

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

induced in a forward stagewise manner (Algorithm 10.2)[2]

At each step we must solve:

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

for $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ given current model $f_{m-1}(x)$.

Given regions R_{jm} finding the optimal constants γ_{jm} is straightforward:

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

Finding the regions is difficult, but in special cases it can be simplified. For example, for exponential or squared-error loss.

Algorithm 8.2: Boosting for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set
2. For $b = 1, \ldots, B$ repeat:
   - 2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r)
   - 2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   - 2.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

[1]

For squared-error loss, the solution is the regression tree that predicts the current residuals $y_i - f_{m-1}(x_i)$ and so is the approach outlined in Algorithm 8.2.

For two-class classification and exponential loss, the stagewise approach gives rise to the AdaBoost method for boosting classification trees in Algorithm 10.1.

Numerical Optimization

The loss in using $f(x)$ to predict y on the training data is

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

To minimize the loss with respect to $f(x)$, this can be viewed as a numerical optimization

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

The goal is to minimize loss, where $f(x)$ is constrained to being a sum of trees.

In this minimization, the $\mathbf{f} \in \mathbb{R}^N$ are the values of the approximating function $f(x_i)$ at each N data points x_i:

$$\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T$$

Numerical Optimization procedures solve this as a sum of component vectors

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

where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess

Steepest Descent

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

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

The *step length* ρ_m is the solution to

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

The solution is then updated

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

Gradient Boosting

The solution in the stagewise approach is analogous to the line search in steepest descent.

If the goal was to simply minimize the loss over the training data, then the steepest descent would work best.

Generalizing the gradient for all data is tricky

A possible resolution is to fit a tree whose predictions are close to the negative gradient.

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:
   - 2.1 For $i = 1, s, \ldots, N$ compute

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

   - 2.2 Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   - 2.3 For $j = 1, 2, \ldots, J_m$ compute

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

   - 2.4 Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}_{\{x \in R_{jm}\}}$
3. Output $\hat{f}(x) = f_M(x)$.

[2]

[1] Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Number v. 6. Springer, 2013.

[2] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Stastical Learning: Data Mining, Inference, and Prediction*. Number v.2 in Springer Series in Statistics. Springer, 2009.