Boosting: Successive Approximation in SML

Stephen Vardeman

Analytics Iowa LLC

ISU Statistics and IMSE

Boosting in predictor development

Boosting is another line of thinking that leads to the use of weighted linear combinations of predictors. This methodology is really just an instance of the basic numerical analysis notion of **successive approximation** to find a solution to an equation or an optimizer of a functional.

There is general gradient boosting (and the famous AdaBoost.M1 algorithm). But we begin with the special case of SEL boosting, because this version is both particularly easy to understand and explain and of high practical value. The basic idea is to repeatedly try to improve an approximator for $\mathrm{E}[y|\mathbf{x}]$ by successively adding small corrections (based on modeling current residuals) to current approximators.

SEL boosting

SEL boosting begins with some predictor $\hat{f}_0(\mathbf{x})$ (like, e.g., $\hat{f}_0(\mathbf{x}) = \bar{y}$). With an iterate $\hat{f}_{m-1}(\mathbf{x})$ in hand, one fits some SEL predictor, say $\hat{e}_m(\mathbf{x})$, to the N "data pairs" $\left(\mathbf{x}_i, y_i - \hat{f}_{m-1}(\mathbf{x}_i)\right)$ consisting of inputs and current residuals. (Typically, some very simple/crude/non-complex predictor form is used for $\hat{e}_m$.) Then, for some "learning rate" $\nu \in (0,1)$, one sets

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \hat{e}_m(\mathbf{x})$$

One iterates on m through some number of iterations, M (typically chosen by cross-validation). Commonly quoted choices for ν are numbers like .01 and the smaller is ν, the larger must be M.

SEL boosting

SEL boosting successively corrects a current predictor by adding to it some fraction of a predictor for its residuals. The value ν functions as a complexity or regularizing parameter, as does M. (Small ν together with large M correspond to large complexity.) Boosting ends with a linear combination of fitted forms as a final predictor/approximator for $\mathrm{E}[y|\mathbf{x}]$.

Sequential modification of a predictor is not discussed in ordinary regression/linear models contexts because if a base predictor is an OLS predictor for a fixed linear model, corrections to an initial fit based on this same model fit to residuals will predict that all residuals are 0. In this circumstance boosting does nothing to change or improve an initial OLS fit.

General gradient boosting

Now consider approximate empirical optimization (over choice of real-valued function g) of

$$\mathrm{E}L\left(g\left(\mathbf{x}\right),y\right)$$

through (successive approximation) search for predictor $\hat{f}$ that optimizes

$$\sum_{i=1}^{N} L\left(\hat{f}\left(\mathbf{x}_i\right),y_i\right) = N\cdot\overline{\mathrm{err}}$$

One begins with some $\hat{f}_0\left(\mathbf{x}\right)$ (like, e.g., $\hat{f}_0\left(\mathbf{x}\right) = \arg\min_{\widehat{y}} \sum_{i=1}^{N} L\left(\widehat{y},y_i\right)$).

With iterate $\hat{f}_{m-1}\left(\mathbf{x}\right)$ in hand, consider how to improve the current total training set loss $\sum_{i=1}^{N} L\left(\hat{f}_{m-1}\left(\mathbf{x}_i\right),y_i\right)$.

Gradient boosting up-dates

Let

$$\widetilde{y}_{im} = -\left.\frac{\partial}{\partial \widehat{y}} L\left(\widehat{y}, y_i\right)\right|_{\widehat{y}=\hat{f}_{m-1}(\mathbf{x}_i)}$$

be the elements of the negative gradient of total loss wrt the training set predictions. One would like to correct $\hat{f}_{m-1}(\mathbf{x})$ in a way that moves each $\hat{f}_{m-1}(\mathbf{x}_i)$ by roughly a common multiple of its $\widetilde{y}_{im}$. To that end, one fits some SEL predictor, say $\hat{e}_m(\mathbf{x})$, to "data pairs" $(\mathbf{x}_i, \widetilde{y}_{im})$. (Typically some very simple form of "base predictor" is used for $\hat{e}_m$.)

Then let $\rho_m > 0$ (controlling the "step-size" in modifying $\hat{f}_{m-1}(\mathbf{x})$) stand for a multiplier for $\hat{e}_m(\mathbf{x})$ such that

$$\sum_{i=1}^{N} L\left(\hat{f}_{m-1}(\mathbf{x}_i) + \rho_m \hat{e}_m(\mathbf{x}_i), y_i\right)$$

is small (ideally, minimum).

Gradient boosting updates cont.

Finally, for some "learning rate" $\nu \in (0, 1)$,

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \rho_m \hat{e}_m(\mathbf{x})$$

is an approximate "steepest descent" correction of $\hat{f}_{m-1}(\mathbf{x})$. (Of course, other criteria besides SEL–like AEL–could be used in fitting $\hat{e}_m(\mathbf{x})$ and the learning rate could be chosen to depend upon m.)

Gradient boosting with trees

The development here allows for arbitrary base predictors. But especially because trees are invariant to monotone transformations of coordinates of $\mathbf{x}$, the functions $\hat{e}_m$ are often rectangle-based (and even restricted to single-split-trees in the case of AdaBoost.M1). If a tree-building applied to "data pairs" $(\mathbf{x}_i, \widetilde{y}_{im})$ produces a set of non-overlapping rectangles $R_1, R_2, \ldots, R_L$ that cover the input space, rather than using for $\hat{e}_m(\mathbf{x})$ in rectangle R_l some average of the values $\widetilde{y}_{im}$ (for training cases with $\mathbf{x}_i \in R_l$) it makes sense to use

$$\hat{e}_m(\mathbf{x}) = \arg\min_c \sum_{i \text{ s.t. } \mathbf{x}_i \in R_l} L\left(\hat{f}_{m-1}(\mathbf{x}_i) + c, y_i\right) \quad \text{for} \quad \mathbf{x} \in R_l$$

and take $\rho_m = 1$. This is the form typically used in gradient boosting with trees.

SEL gradient boosting

We looked first SEL boosting. To establish that it is a version of gradient boosting, simply now suppose that $L(\widehat{y}, y) = \frac{1}{2}(\widehat{y} - y)^2$. Then

$$\widetilde{y}_{im} = -\frac{\partial}{\partial \widehat{y}}\left(\frac{1}{2}(\widehat{y} - y_i)^2\right)\Big|_{\widehat{y}=f_{m-1}(\mathbf{x}_i)} = y_i - \hat{f}_{m-1}(\mathbf{x}_i)$$

and for SEL the general gradient boosting corrections are indeed based on the prediction of ordinary residuals.

AEL gradient boosting

Suppose next that $L(\widehat{y}, y) = |\widehat{y} - y|$. Then, beginning from $\hat{f}_0(\mathbf{x})$ (say $\hat{f}_0(\mathbf{x}) = \text{median}\{y_i\}$),

$$\widetilde{y}_{im} = -\frac{\partial}{\partial \widehat{y}}\left(|\widehat{y} - y_i|\right)\Big|_{\widehat{y}=\hat{f}_{m-1}(\mathbf{x}_i)} = \text{sign}\left(y_i - \hat{f}_{m-1}(\mathbf{x}_i)\right)$$

So a gradient boosting update step is "fit a SEL predictor for ± 1s coding the signs of the residuals from the previous iteration." In the event that the base predictors are regression trees, $\hat{e}_m(\mathbf{x})$ in a rectangle will be a median of ± 1s coming from signs of residuals for cases with $\mathbf{x}_i$ in the rectangle (and thus have value either -1 or 1, constant on the rectangle).

Binomial deviance loss gradient boosting

Referring again to our development of optimal voting functions for 2-class classifiers, it's clear that approximation to optimal voting functions $g(\mathbf{x})$ can produce approximately optimal 2-class classifiers. Then consider $h_1(u) = \ln(1+\exp(-u)) / \ln(2)$ and the loss

$$L(g(\mathbf{x}), y) = h_1(yg(\mathbf{x})) = \ln(1+\exp(-yg(\mathbf{x}))) / \ln(2)$$

For this situation

$$\begin{aligned}\widetilde{y}_{im} &= -\frac{\partial}{\partial \widehat{y}} \left(\ln(1+\exp(-y_i\hat{y})) / \ln(2)\right)\Big|_{\widehat{y}=\hat{f}_{m-1}(\mathbf{x}_i)} \\ &= \frac{1}{\ln 2}\left(\frac{\hat{f}_{m-1}(\mathbf{x}_i)\exp(-y_i\hat{y}_i)}{1+\exp\left(-y_i\hat{f}_{m-1}(\mathbf{x}_i)\right)}\right)\end{aligned}$$

and boosting can be expected to produce a voting function approximating the log likelihood ratio.

Exponential loss gradient boosting

For the exponential function $h_2(u) = \exp(-u)$ and 2-class classification loss $L(g(\mathbf{x}), y) = h_2(yg(\mathbf{x}))$ one has

$$\widetilde{y}_{im} = -\frac{\partial}{\partial \widehat{y}} \exp(-y_i \widehat{y}) \Big|_{\widehat{y}=\hat{f}_{m-1}(\mathbf{x}_i)} = y_i \exp\left(-y_i \hat{f}_{m-1}(\mathbf{x}_i)\right)$$

and boosting produces a voting function approximating half of the log likelihood ratio. (For the choice of base predictors as single-split trees, gradient boosting is a version of the famous AdaBoost.M1 algorithm.)

Hinge loss gradient boosting

For the "hinge" function $h_3(u) = (1-u)_+$ and 2-class classification loss $L(g(\mathbf{x}), y) = h_3(yg(\mathbf{x}))$, one gets

$$\widetilde{y}_{im} = -\frac{\partial}{\partial \widehat{y}}(1 - y_i\hat{y})_+\Big|_{\widehat{y}=\hat{f}_{m-1}(\mathbf{x}_i)} = y_i I\left[y_i \hat{f}_{m-1}(\mathbf{x}_i) < 1\right]$$

and boosting produces a voting function approximating the optimal classifier directly.

Boosting in practice: tree complexity

There are several issues that arise in the practical use of boosting, particularly if trees are the base predictors. These mostly concern control of complexity parameters/avoiding over-fit/regularization in boosting.

One such issue is the question of how large trees should be allowed to grow if they are used as the functions $\hat{e}_m(\mathbf{x})$ in a boosting algorithm. The answer seems to be "Not too large, maybe to about 6 or so terminal nodes." Another (probably better) approach to this question would seem to be to grow large trees and then employ cost-complexity pruning, ultimately using cross-validation to choose a value for the weight α (or $\lambda = 1/\alpha$).

Boosting in practice: M and shrinkage

M can/should be limited in size (very large values surely producing over-fit). A crude methodology suggested early in the development of boosting was to hold back a part of a training sample and watch performance of a predictor on that single test set as M increases.

"Shrinkage"/use of $\nu \in (0, 1)$ and

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \rho_m \hat{e}_m(\mathbf{x})$$

provides regularization in gradient boosting (as for the SEL case). This doesn't make the "full correction" to $\hat{f}_{m-1}$ in producing $\hat{f}_m$ and typically requires larger M for good prediction than the non-shrunken version.

Boosting in practice: subsampling and CV

The notion of "subsampling" in boosting is that at each iteration of a boosting program, instead of choosing an update based on the whole training set, one chooses a fraction η of the training set at random, and fits to it (using a new random selection at each iteration). This methodology might be called "stochastic gradient boosting," reduces the computation time per iteration, and can improve predictor performance.

Ultimately, all of 1) control of parameters governing tree size, 2) choice of M and ν, and 3) choice of parameters controlling details of subsampling can be subjected to (joint) optimization/tuning using cross-validation.

Boosting in practice: XGBoost

A very popular implementation of gradient boosting goes by the name "XGBoost" (for "eXtreme Gradient Boosting"). This is an R package (with similar implementations in other systems) that provides a lot of flexibility in the implementation of boosting, and code that is very fast to run (even providing parallelization where hardware supports it). The caret package can be used to do cross-validation for XGBoost, allowing one to tune on a number of algorithm complexity parameters.