Boosting

Like bagging and random forests, boosting combines multiple weak learners into one improved model. Unlike bagging and random forests, however, boosting trains these weak learners *sequentially*, each one learning from the mistakes of the last. Rather than a single model, "boosting" refers to a class of sequential learning methods.

Here we discuss two specific algorithms for conducting boosting. The first, known as *Discrete AdaBoost* (or just *AdaBoost*), is used for binary classification. The second, known as *AdaBoost.R2*, is used for regression.

AdaBoost for Binary Classification

Weighted Classification Trees

Weak learners in a boosting model learn from the mistakes of previous iterations by increasing the *weights* of observations that previous learners struggled with. How exactly we fit a *weighted* learner depends on the type of learner. Fortunately, for classification trees this can be done with just a slight adjustment to the loss function.

Consider a classification tree with classes $k = 1, \dots, K$ and a node $\mathcal{N}_m$. Let $\hat{p}_{mk}$ give the fraction of the observations in $\mathcal{N}_m$ belonging to class k. Recall that the Gini index is defined as

$$\mathcal{L}_G(\mathcal{N}_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

and the cross entropy as

$$\mathcal{L}_E(\mathcal{N}_m) = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk},$$

The classification tree then chooses the split S_m that minimizes the combined loss of the child nodes,

$$\mathcal{L}(S_m) = f_L \cdot \mathcal{L}(\mathcal{C}_m^L) + f_R \cdot \mathcal{L}(\mathcal{C}_m^R),$$

where C_m^L and C_m^R are the left and right child nodes and f_L and f_R are the fraction of observations going to each.

To allow for observation weighting, we simply change our definition of $\hat{p}_{mk}$ to the following:

$$\hat{p}_{mk} = \frac{\sum_{n \in \mathcal{N}_m} w_n I(y_n = k)}{\sum_{n \in \mathcal{N}_m} w_n}.$$

This forces the tree to prioritize isolating the classes that were poorly classified by previous learners. The Gini index and cross-entropy are then defined as before and the rest of the tree is fit in the same way.

The Discrete AdaBoost Algorithm

The *Discrete AdaBoost* algorithm is outlined below. We start by setting the weights equal for each observation. Then we build T weighted weak learners (decision trees in our case). Each observation's weight is updated according to two factors: the strength of the iteration's learner and the accuracy of the learner on that observation. Specifically, the update for the weight on observation n at iteration t takes the form

$$w_n^{t+1} = w_n \exp(\alpha^t I_n^t),$$

where α^t is a measure of the accuracy of the learner and I_n^t indicates the learner misclassified observation n. This implies that the weight for a correctly-classified observation does not change while the weight for a misclassified observation increases, particularly if the model is accurate.

> **Note**
>
> Notation for boosting can get confusing since we iterate over learners and through observations. For the rest of this page, let superscripts refer to the model iteration and subscripts refer to the observation. For instance, w_n^t refers to the weight of observation n in learner t.

Discrete AdaBoost Algorithm

Define the target variable to be $y_n \in \{-1, +1\}$.

1. Initialize the weights with $w_n^1 = \frac{1}{N}$ for $n = 1, 2, \dots, N$.

2. For $t = 1, \ldots, T$,
    - Build weak learner t using weights $\mathbf{w}^t$.
    - Use weak learner t to calculate fitted values $f^t(\mathbf{x}_n) \in \{-1, +1\}$ for $n = 1, 2, \ldots, N$. Let I_n^t equal 1 If $f^t(\mathbf{x}_n) \neq y_n$ and 0 otherwise. That is, I_n^t indicates whether learner t misclassifies observation n.
    - Calculate the weighted error for learner t:

$$\epsilon^t = \frac{\sum_{n=1}^N w_n^t I_n^t}{\sum_{n=1}^N w_n^t}.$$

    - Calculate the accuracy measure for learner t:

$$\alpha^t = \log\left(\frac{1 - \epsilon^t}{\epsilon^t}\right).$$

    - Update the weighting with

$$w_n^{t+1} = w_n^t \exp(\alpha^t I_n^t),$$

    for $n = 1, 2, \ldots, N$.
3. Calculate the overall fitted values with $\hat{y}_n = \text{sign}\left(\sum_{t=1}^T \alpha^t f^t(\mathbf{x}_n)\right)$.

Note that our final fitted value for observation n, $\hat{y}_n$, is a weighted vote of all the individual fitted values where higher-performing learners are given more weight.

Print to PDF

AdaBoost For Regression

We can apply the same principle of learning from our mistakes to regression tasks as well. A common algorithm for doing so is *AdaBoost.R2*, shown below. Like *AdaBoost*, this algorithm uses weights to emphasize observations that previous learners struggled with. Unlike *AdaBoost*, however, it does not incorporate these weights into the loss function directly. Instead, every iteration, it draws bootstrap samples from the training data where observations with greater weights are more likely to be drawn.

We then fit a weak learner to the bootstrapped sample, calculate the fitted values on the *original sample* (i.e. *not* the bootstrapped sample), and use the residuals to assess the quality of the weak learner. As in *AdaBoost*, we update the weights each iteration based on the quality of the learner (determined by β^t) and the accuracy of the learner on the observation (determined by L_n).

After iterating through many weak learners, we form our fitted values. Specifically, for each observation we use the weighted median (defined below) of the weak learners' predictions, weighted by $\log(1/\beta^t)$ for $t = 1, \ldots, T$.

> **Note**
>
> **Weighted Median**
>
> Consider a set of values $\{v_1, v_2, \ldots, v_N\}$ and a set of corresponding weights $\{s_1, s_2, \ldots, s_N\}$. Normalize the weights so they add to 1. To calculate the weighted median, sort the values and weights in ascending order of the values; call these $\{v^{(1)}, v^{(2)}, \ldots, v^{(N)}\}$ and $\{s^{(1)}, s^{(2)}, \ldots, s^{(N)}\}$ where, for instance, $v^{(1)}$ is the smallest value and $s^{(1)}$ is its corresponding weight. The weighted median is then the value corresponding to the first weight such that the cumulative sum of the weights is greater than or equal to 0.5.
>
> As an example, consider the values $\mathbf{v} = \{10, 30, 20, 40\}$ and corresponding weights $\mathbf{s} = \{0.4, 0.6, 0.2, 0.8\}$. First normalize the weights to sum to 1, giving $\mathbf{s} = \{0.2, 0.3, 0.1, 0.4\}$. Then sort the values and the weights, giving $\mathbf{v} = \{10, 20, 30, 40\}$ and $\mathbf{s} = \{0.2, 0.1, 0.3, 0.4\}$. Then, since the third element is the first for which the cumulative sum of the weights is at least 0.5, the weighted median is the third of the sorted values, 30.

Though the arithmetic below gets somewhat messy, the concept is simple: iteratively fit a weak learner, see where the learner struggles, and emphasize the observations where it failed (where the amount of emphasis depends on the overall strength of the learner).

AdaBoost.R2 Algorithm

1. Initialize the weights with $w_n^1 = \frac{1}{N}$ for $n = 1, 2, \ldots, N$.
2. For $t = 1, 2, \ldots, T$ or while $\bar{L}^t$, as defined below, is less than or equal to 0.5,
    - Draw a sample of size N from the training data with replacement and with probability w_n^t for $n = 1, 2, \ldots, N$.
    - Fit weak learner t to the resampled data and calculate the fitted values on the original dataset. Denote these fitted values with $f^t(\mathbf{x}_n)$ for $n = 1, 2, \ldots, N$.
    - Calculate the observation error L_n^t for $n = 1, 2, \ldots, N$:

$$D^t = \max_n\{|y_n - f^t(\mathbf{x}_n)|\}$$
$$L_n^t = \frac{|y_n - f^t(\mathbf{x}_n)|}{D^t}$$

- Calculate the model error $\bar{L}^t$:

$$\bar{L}^t = \sum_{n=1}^{N} L_n^t w_n^t$$

  If $\bar{L}^t \geq 0.5$, end iteration and set T equal to $t-1$.
- Let $\beta^t = \frac{\bar{L}^t}{1-\bar{L}^t}$. The lower β^t, the greater our confidence in the model.
- Let $Z^t = \sum_{n=1}^{N} w_n^t (\beta^t)^{1-L_n^t}$ be a normalizing constant and update the model weights with

$$w_n^{t+1} = \frac{w_n^t (\beta^t)^{1-L_n^t}}{Z^t},$$

  which increases the weight for observations with a greater error L_n^t.

3. Set the overall fitted value for observation n equal to the weighted median of $f^t(\mathbf{x}_n)$ for $t = 1, 2, \ldots, T$ using weights $\log(1/\beta^t)$ for model t.

In the boosting construction, we implement *AdaBoost.R2* using decision tree regressors though many other weak learners may be used.

By Danny Friedman