ECON 950 — Winter 2020

Prof. James MacKinnon

7. Boosting

Like bagging and random forests, boosting involves creating many models.

Unlike bagging and random forests, boosting creates these models sequentially, and there is no resampling involved.

Boosting **learns slowly** by repeatedly fitting new models based on the residuals of earlier models.

For regression trees, the algorithm works as follows:

1. Set $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \mathbf{0}$ and $r_i = y_i$ for all i.
2. For $m = 1, \ldots, M$, repeat:
   i. Fit a tree $\hat{\boldsymbol{f}}^m(\boldsymbol{x})$ with d splits to the training data $(\boldsymbol{x}, \boldsymbol{r})$.
   ii. Update $\hat{\boldsymbol{f}}$ by adding a shrunken version of $\hat{\boldsymbol{f}}^m(\boldsymbol{x})$ to it:

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) \leftarrow \hat{\boldsymbol{f}}(\boldsymbol{x}) + \lambda \hat{\boldsymbol{f}}^m(\boldsymbol{x}). \tag{1}$$

iii. Update the residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^m(\boldsymbol{x}_i). \tag{2}$$

3. The boosted model is

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \sum_{m=1}^{M} \lambda \hat{\boldsymbol{f}}^m(\boldsymbol{x}). \tag{3}$$

Recall that the initial value of $\hat{\boldsymbol{f}}$ was $\mathbf{0}$. Thus all of the explanatory power comes from the $\hat{\boldsymbol{f}}^m(\boldsymbol{x})$.

There are three tuning parameters:

1. The number of (in this case) trees. There is a risk of overfitting if M gets too large, so we need to use cross-validation.
2. The **shrinkage parameter** λ. Typical values are 0.01 and 0.001. When λ is very small, M needs to be large.
3. The number of splits d, called the **interaction depth**. This tends to be small, perhaps just $d = 1$.

When $d = 1$, every split is a stump, and (3) becomes an additive model.

It seems odd that λ is not set to $1/M$. Do we have to rescale λ if M becomes larger?

See ISLR Figure 8.11.

For squared error loss (which may not be a good thing to use; see ESL Section 10.6), the objective function is

$$\sum_{i=1}^{N} L\big(y_i, f(\boldsymbol{x}_i)\big) = \sum_{i=1}^{N} \big(y_i - f(\boldsymbol{x}_i)\big)^2. \tag{4}$$

Since we build up the $f(\boldsymbol{x}_i)$ slowly in (1), we see that

$$\begin{aligned} L\big(y_i, \hat{f}(\boldsymbol{x}_i)\big) &\leftarrow L\big(y_i, \hat{f}(\boldsymbol{x}_i) + \lambda \hat{f}^m(\boldsymbol{x}_i)\big) \\ &= \big(y_i - \hat{f}(\boldsymbol{x}_i) - \lambda \hat{f}^m(\boldsymbol{x}_i)\big)^2 \\ &= \big(r_i - \lambda \hat{f}^m(\boldsymbol{x}_i)\big)^2, \end{aligned} \tag{5}$$

where r_i is simply the i^{th} residual for the current model, before we have added the m^{th} term to it.

At each step in the boosting algorithm, we add λ times the term $\hat{f}^m(\boldsymbol{x}_i)$ that best fits the function

$$\sum_{i=1}^{N} \big(r_i - \hat{f}^m(\boldsymbol{x}_i)\big)^2, \tag{6}$$

which just depends on the current residuals and on $\hat{f}^m(\boldsymbol{x}_i)$.

The discussion above assumes that we use a tree to obtain $\hat{f}^m(\boldsymbol{x}_i)$, but many other models can also be boosted.

It does not make sense to boost a linear regression model, because the residuals are orthogonal to all the predictors.

7.1. AdaBoost.M1 for two-way classification

We code the output as $\{-1, 1\}$ instead of $\{0, 1\}$.

The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\big(y_i \neq G(\boldsymbol{x}_i)\big), \tag{7}$$

where $G(\boldsymbol{x}_i)$ denotes the classifier. It also takes values $\{-1, 1\}$.

A **weak classifier** is only slightly better than random guessing.

Boosting sequentially applies the weak classifier to repeatedly modified versions of the data (in the regression case above, these were residuals). Eventually, we have M of these, denoted $G_m(\boldsymbol{x})$ for $m = 1, \ldots, M$.

We then combine these as follows to produce the final prediction

$$G(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right), \tag{8}$$

where the weights α_m have to be determined.

Instead of using residuals from successive steps, classification uses weights that change as the algorithm proceeds.

Observations that were misclassified get more weight, and observations that were correctly classified get less weight.

The `AdaBoost.M1` algorithm works as follows:

1. Initialize the observations weights to $1/N$ for all i.
2. For $m = 1, \ldots, M$:
   i. Fit a classifier $G_m(\boldsymbol{x})$ to the training data using weights w_i.
   ii. Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\big(y_i \neq G_m(\boldsymbol{x}_i)\big)}{\sum_{i=1}^{N} w_i}.$$
   iii. Compute $\alpha_m = \log\big((1 - \text{err}_m)/\text{err}_m\big)$.
   iv. Set $w_i \leftarrow w_i \exp\Big(\alpha_m \mathbb{I}\big(y_i \neq G_m(\boldsymbol{x}_i)\big)\Big)$ for $i = 1, \ldots, N$.
3. The final classifier is given in (8): $G(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$.

The algorithm just described is called "Discrete AdaBoost," because the classifier always reports -1 or 1.

A modified version called "Real AdaBoost" was proposed in Friedman, Hastie, and Tibshirani (*AMS*, 2000). This seems to be a key paper. Real AdaBoost yields real numbers in the $[0, 1]$ interval as outputs instead of $\{-1, 1\}$.

In the above algorithm, step 1. is unchanged. Step 2. becomes

2. For $m = 1, \ldots, M$:
   i. Fit a classifier $p_m(\boldsymbol{x})$ to the training data using weights w_i.
   ii. Compute $f_m = \frac{1}{2} \log\big(p_m(\boldsymbol{x})/(1 - p_m(\boldsymbol{x}))\big)$.
   iii. Set $w_i \leftarrow w_i \exp\Big(-y_i f_m(\boldsymbol{x}_i)\Big)$ for all i, and renormalize so that $\sum_{i=1}^{N} w_i = 1$.
3. The final classifier is the sign of $\sum_{m=1}^{M} f_m(\boldsymbol{x})$.

Observe that the f_m here are the logs of odds ratios.

ESL present a simulated example in which $p = 10$ and each input X_j is $\mathrm{N}(0, 1)$.

The output is 1 if

$$\sum_{j=1}^{10} X_j^2 > \chi^2_{10}(0.5) = 9.34182. \tag{9}$$

Thus, on average, half of the outputs will be 1 and half will be -1. But, since we observe the realized random variables, we should be able to make predictions.

There are 2000 training cases and 10,000 test ones.

The weak classifier is a tree with two terminal nodes, a "stump."

The error rate of the initial classifier is 45.8%. After boosting with $M = 400$, it is only 5.8%. A single large tree has an error rate of 24.7%.

See ESL, Figure 10.2.

7.2. Boosting and additive models

Because boosting is additive — see (3) and (8) — it can be thought of as a form of basis expansion.

Basis expansions take the form

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}, \boldsymbol{\gamma}_m), \tag{10}$$

where the β_m are expansion coefficients, and the $b(\cdot)$ are (usually) simple functions of $\boldsymbol{x}$ and the coefficient vectors $\boldsymbol{\gamma}_m$.

For neural nets, the $b(\cdot)$ are sigmoid functions of linear combinations of $\boldsymbol{x}$.

For MARS, the $b(\cdot)$ are splines, and the $\boldsymbol{\gamma}_m$ parametrize the variables and the values for the knots.

For trees, the $\boldsymbol{\gamma}_m$ parametrize the split variables and split points.

In general, we estimate this sort of model by minimizing a loss function with respect to the β_m and the $\boldsymbol{\gamma}_m$:

$$\sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}, \boldsymbol{\gamma}_m)\right) \tag{11}$$

This is hard if we have to minimize with respect to all parameters at once, but it can be easy if we can minimize sequentially with respect to the parameters of one basis function at a time.

The idea of **forward stagewise additive modeling** is to minimize (11) by adding additional basis functions *without* modifying previous ones.

This is exactly what boosting does; see (3) and the algorithm that includes it.

7.3. Other issues in boosting

What does AdaBoost minimize?

According to ESL, it minimizes

$$L\big(y, f(\boldsymbol{x})\big) = \exp\big(-y\, f(\boldsymbol{x})\big); \tag{12}$$

see Section 10.4.

They show that this is equivalent to minimizing the deviance

$$\ell\big(y, f(\boldsymbol{x})\big) = \log\big(1 + \exp(-2y\, f(\boldsymbol{x}))\big), \tag{13}$$

which is what a logit model would minimize, except for the factor of 2 that arises because of coding the output as $\{-1, 1\}$ instead of $\{0, 1\}$.

There is a long discussion of robust and non-robust loss functions in Section 10.6.

There is a detailed discussion of boosting trees in Section 10.9.

The important topic of **gradient boosting** is discussed in Section 10.10.

There are some interesting examples in Section 10.14.