36-724 Spring 2006: Bagging and Boosting; Application

Brian Junker

April 7, 2006

- Introduction
- Bagging
- Boosting
- Comments and Tentative Conclusions
Introduction

Trees are an example of a prediction method that is intuitively appealing (simple/sparse structure, easy to interpret/implement) but does not produce great error rates.

- High variance because the partition elements $R_m$ depend on the configuration of the training data.
- High bias when the action of the “true” function does not line up with the coordinate axes.

Each is a weakness that can contribute to poor error rates with new data.
Ensemble learning or committee-based methods have been developed to deal with each of these problems. Two of the most popular are:

- **Bagging:** (B)ootstrap (Ag)gregation. Use the average (majority rule, etc.) of a committee of predictors trained on different bootstrap samples. *Focuses on variance reduction.*

- **Boosting:** Also uses the (weighted) average of a committee of predictors, but the predictors are trained on different transformations of the data (often, each predictor is trained on the residuals of other predictors in the committee). *Focuses on bias reduction.*
Bagging

Bootstrap aggregation (bagging) is based on the idea of averaging to reduce bias.

Let $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ be the original training data and let $S^*_1, \ldots, S^*_B$ be $B$ bootstrap replications (samples of size $N$ with replacement) of the training data.

Fit the prediction function $\hat{f}^{*b}(x)$ to each bootstrap sample; then the bagged prediction function is

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$
**Example: Bagging an under-smoothed regression**

Here the data is the famous motorcycle head-acceleration data; the prediction function is estimated from kernel smoothing; $\hat{f}_{bag}(x)$ is based on 100 bootstrap samples.
Why does it work?

• Bagging tends to improve squared error loss. Let $f_{true}(x)$ be the true value; since $\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{\*b}(x)$, we know:

$$MS E[\hat{f}(x)] = (E[\hat{f}(x)] - f_{true}(x))^2 + \text{Var} [\hat{f}(x)];$$

$$MS E[\hat{f}_{bag}(x)] = (E[\hat{f}_{bag}(x)] - f_{true}(x))^2 + \text{Var} [\hat{f}_{bag}(x)]$$

$$\approx (E[\hat{f}(x)] - f_{true}(x))^2 + \frac{1}{B} \text{Var} [\hat{f}(x)]$$

$$< MS E[\hat{f}(x)]$$

• For 0-1 loss, bagging tends to make unbiased classifiers better, and biased ones worse. *Reason:*

  – An **unbiased** classifier will make errors due to variability, and $\hat{f}_{bag}(x)$ has less variability, so fewer errors.

  – A **biased** classifier makes *errors* due to bias, and some *correct classifications* due to variability. $\hat{f}_{bag}(x)$ has less variability, so fewer correct classifications.
Example: Bagging a tree classifier

$X = (X_1, \ldots, X_5)$ are standard normal, correlated $\rho = 0.95$.

$P[Y = 1|X_1 < 0.5] = 0.2; \ P[Y = 1|X_1 \geq 0.5] = 0.8$. $N = 100$ observations in both training and test samples. Best possible classification error rate is clearly 20%.
Boosting

- Made famous by AdaBoost
- Final prediction is a weighted average of a “committee” of \( M \) predictor functions.
- Each predictor function in the committee depends on its predecessors.
- Training Examples may have unequal weights. Look at errors from previous classifier step to decide how to focus on next iteration over data.
- Set weights to focus more on hard examples (the ones on which we committed mistakes in the previous iterations).
**Boosting as an Additive Model**

The final predictor function $f_M(x)$ is built up iteratively.

Let $L(y, f(x))$ be a loss function, and let $b(x; \gamma)$ be a family of (simple) prediction functions depending on parameters $\gamma$.

\[
f_1(x) = \beta_1 b(x; \gamma_1)
\]

where $(\beta_1, \gamma_1) = \text{argmin}_{\beta, \gamma} \sum_{i=1}^{N} L[y_i, \beta b(x_i; \gamma)]$

\[
f_2(x) = f_1(x) + \beta_2 b(x; \gamma_2) \equiv \beta_1 b(x; \gamma_1) + \beta_2 b(x; \gamma_2)
\]

where $(\beta_2, \gamma_2) = \text{argmin}_{\beta, \gamma} \sum_{i=1}^{N} L[y_i - f_1(x_i), \beta b(x_i; \gamma)]$

\[
\vdots
\]

\[
f_M(x) = f_{M-1}(x) + \beta_M \equiv \sum_{m=1}^{M-1} \beta_m b(x; \gamma_m) + \beta_M b(x; \gamma_M)
\]

where $(\beta_M, \gamma_M) = \text{argmin}_{\beta, \gamma} \sum_{i=1}^{N} L[y_i - f_{M-1}(x_i), \beta b(x_i; \gamma)]$

(this is exactly analogous to how a ppreg model is built up for example!)
Example: Boosting a Piecewise Constant Regression

The data is again the motorcycle head accelleration data. The basic regression function is

\[ b(x; s, \ell, r) = \ell \cdot 1_{\{x<s\}} + r \cdot 1_{\{x\geq s\}} \]

and the boosted regression (squared error loss) after \( M \) iterations is

\[ f_M(x) = \sum_{m=1}^{M} \beta_m \cdot (\ell_m \cdot 1_{\{x<s_m\}} + r_m \cdot 1_{\{x\geq s_m\}}) \]
**AdaBoost**

Squared error not appropriate for classification.

Exponential loss: Code $y = \pm 1$ again, and let $L(y, f(x)) = \exp(-y \cdot f(x))$:

$$
\min_{\beta, \gamma} \sum_{i=1}^{N} L[y_i, f_M(x)]
= \min_{\beta, \gamma} \sum_{i=1}^{N} \exp(-y_i [f_{M-1}(x) + \beta b(x_i; \gamma)])
= \min_{\beta, \gamma} \sum_{i=1}^{N} \exp(-y_i f_{M-1}(x)) \exp(-y \cdot \beta b(x_i; \gamma))
= \min_{\beta, \gamma} \sum_{i=1}^{N} w_i^{M-1} \exp(-y_i \cdot \beta b(x_i; \gamma))
= \min_{\beta, \gamma} \sum_{y_i = b(x; \gamma)} w_i^{M-1} \exp(-\beta) + \sum_{y_i \neq b(x; \gamma)} w_i^{M-1} \exp(\beta)
$$

Note that $w_i^{M-1}$ weights errors more than corrects. For fixed $\beta$ the above is equivalent to minimizing, w.r.t. $\gamma$,

$$
err = \frac{\sum_{y_i \neq b(x; \gamma)} w_i^{M-1}}{\sum_{i=1}^{N} w_i^{M-1}}
$$

and for fixed $\gamma$ one can show that the minimizing $\beta = \frac{1}{2} \log \frac{1 - err}{err}$.
This gives rise to the following algorithm:

- \( W(x) \) is the distribution of weights over the \( N \) training points, with \( \sum_i W(x_i) = 1 \). Initially assign uniform weights \( W_0(x) = 1/N \) for all \( x \).

- At each iteration \( m \):
  - Find best weak classifier \( C_m(x) \) minimizing \( \epsilon_m \), the weighted error rate:
    \[
    \epsilon_m = \frac{\sum_i W_m(x_i) 1_{\{y_i \neq C_m(x_i)\}}}{\sum_i W_m(x_i)}
    \]
  - The classifier \( C_m \)'s gets weight \( \beta_m \) in the final classifier, where
    \[
    \beta_m = \log((1 - \epsilon_m)/\epsilon_m)
    \]
  - For each \( x_i \), \( W_{m+1}(x_i) = W_m(x_i) \exp[\beta_m \cdot 1_{\{y_i \neq C_m(x_i)\}}] \)

- \( C_{FINAL}(x) = \text{sign}[\sum_m a_mC_m(x)] \)
R packages that do boosting


- R package `boost` (install from CRAN) also implements AdaBoost and several boosted regression methods (again based on regression trees). `boost` is not as flexible as `gbm` but is convenient for illustration...

- Boosting has been a very successful nonparametric regression and classification method. Breiman (1996, NIPS Workshop) referred to AdaBoost with trees as the “best off-the-shelf-classifier in the world”.
Bivariate observations in the unit square, those below the line $y = 1 - x$ belong to one class and those above to the other class.

100 training observations; 100 testing observations; $M = 100$ committee member classifiers in the weighted average.

Two different boosting operations tried:

- **Straight AdaBoost**: boosting “tree stump” classifiers, that is, classifiers that make one cut on one axis only, as the “committee members”

- **“l2boost”**: boosting logistic regressions for 0/1 outcomes as the “committee members”

- **Note**: These are probably not very fair tests of boosting simple classifiers, which seem to require lots and lots of training data and very large committees of simple classifiers...
Adaboost results

Adaboost

Boosting steps

Error rate
**l2boost results**

![L2Boost Graph]

- **X-axis:** Boosting steps
- **Y-axis:** Error rate

The graph shows the behavior of the l2boost algorithm over boosting steps, with error rate on the y-axis and boosting steps on the x-axis. The horizontal line at the end suggests stabilization of the error rate.
Comments and Tentative Conclusions

- The exponential loss function leads to
  - Weights $W_k(x)$ easy to update from one iteration to the next
  - Easy derivatives for maximization with respect to $\beta$
- In a two-class problem AdaBoost determines $\frac{1}{2} \log \frac{P[Y=1|x]}{1-P[Y=1|x]}$
- Need a lot of data
- Need individual classifiers which are either unbiased or disagree a lot.
- Can be extended to trees
- Like Bagging, Boosting can greatly improve error rates of trees, but at the expense of simple interpretability/implementation.