Data Mining and Machine Learning, Boosting, Mar 30, 2011

Boosting – Overview

- The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers(slightly better than random guess) to produce a powerful "committee".

- Consider a two class problem, with the output coded as $Y \in \{-1, 1\}$, predictor variables X, a classifier $G(X)$. The error rate is

$$err = \frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq G(x_i)\right)$$

- The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifier $G_m(x) = 1, 2, ..., M$. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction:

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

  Here $\alpha_1, \alpha_2, ..., \alpha_M$ are computed by the boosting algorithm and weigh the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence. (Fig 10.1)

- AdaBoost.M1.

```
1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, ..., N$.
2. For $m = 1$ to $M$ :
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$err_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m(x_i)\right)}{\sum_{i=1}^{N} w_i}$$
    (c) Compute $\alpha_m = \log\left((1 - err_m)/err_m\right)$.
    (d) Set $w_i \leftarrow w_i \exp\left[\alpha_m I\left(y_i \neq G_m(x_i)\right)\right]$, $i = 1, 2, ..., N$.
3. Output $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
```

Boosting Fits an Additive Model

- The key to the success of boosting lies in the its additive expansion in a set of elementary "basis". Here the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. Generally basis function expansions take the form

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

  where $\beta_m, m = 1, 2, ..., M$ are the expansion coefficients

- Many learning methods have additive basis expansions.
  - MARS uses truncated power spline basis functions where γ parametrizes the variables and values for the knots
  - For trees, basis functions are step functions and γ parametrizes the split variables and split points and the predictions at the terminal nodes.

- Typically these methods are fit by minimizing loss function average over the training data, such as squared-error or a likelihood-based loss function

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

- For many loss functions $L(y, f(x))$ and/or basis functions $b(x; \gamma)$ this requires computationally intensive numerical optimization techniques. However, a simple alternative often can be found when it is feasible to solve the subproblem of fitting just a single basis function,

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

Forward Stagewise Additive Modeling

Forward stagewise modeling approximates the solution to additive models by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added.

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

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

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

Exponential Loss and AdaBoost

- AdaBoost.M1 is equivalent to forward stagewise additive modeling using the loss function
$$L\left(y, f\left(x\right)\right) = \exp\left(-yf\left(x\right)\right).$$

- For AdaBoost the basis functions are individual classifiers $G_m\left(x\right) \in \{-1, 1\}$. Using the exponential loss function, one must solve
$$\left(\beta_m, G_m\right) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp\left[-y_i(f_{m-1}\left(x_i\right) + \beta G\left(x_i\right))\right]$$
for the classifier G_m and corresponding coefficient β_m to be added to each step. This can be expressed as
$$\left(\beta_m, G_m\right) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-y_i \beta G\left(x_i\right)\right)$$
with $w_i^{(m)} = \exp\left(-y_i f_{m-1}\left(x_i\right)\right)$. Since each $w_i^{(m)}$ depends neither on β nor $G\left(x\right)$, it can be regraded as a weight that is applied to each observation. This weight depends on $f_{m-1}\left(x_i\right)$, and so the individual weight values change with each iteration m.

- The solution for $\left(\beta_m, G_m\right)$ can be obtained in two steps. First, for any value of $\beta > 0$, the solution for $G_m\left(x\right)$ is
$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G\left(x_i\right)\right),$$
which is the classifier that minimizes the weighted error rate in predicting y. Plugging this G_m and solve for β, one obtains
$$\beta_m = \frac{1}{2} \log \frac{1 - err_m}{err_m},$$
where err_m is the minimized weighted error rate
$$err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i^{(m)}}$$

- The approximation is then updated

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

  which causes the weights for the next iteration to be

  $$w_i^{(m+1)} = w_i^{(m)} e^{-\beta_m y_i G_m(x)}$$

  Using the fact that $-y_i G_m(x_i) = 2I(y_i \neq G_m(x_i)) - 1$,

  $$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(y_i \neq G_m(x_i))} e^{-\beta_m}$$

  where $\alpha_m = 2\beta_m$ is the quantity defined at line 2c of AdaBoost.M1. The factor $e^{-\beta_m}$ multiples all weights by the same value, so it has no effect. Thus the weights updating is equivalent to line 2d of AdaBoost.M1. One can view line 2a of AdaBoost.M1 as a method of solving G_m. Hence we conclude that AdaBoost.M1 minimizes the exponential loss criterion via a forward-stagewise additive modeling approach.

The Exponential Loss

- It is easy to show that

  $$f^*(x) = \arg\min_{f(x)} E_{Y|x}\left(e^{-Yf(x)}\right) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

  or equivalently

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

  This justifies using the sign of the classification function as the rule.

- The exponential loss is monotone decreasing functions of the "margin" $yf(x)$. In classification with a -1/1 response, the margin plays a role analogous to the residuals $y - f(x)$ in regression. The goal of the classification algorithm is to produce positive margins as frequently as possible. Any loss criterion used for classification should penalize negative margins more heavily than positive ones.

- Fig 10.4 shows some loss functions as a function of the margin $yf(x)$.

  - At any point in the training process the exponential loss concentrates much more influence on observations with large negative margins. Binomial deviance more evenly spread the influence among all of the data. The latter is therefore far more robust in the noisy settings, and especially in the situations where there is misspecification of the data labels in the training data.
  - Square-error loss is not a good surrogate for misclassification error. It increases quadratically for $y_i f(x_i)$, thereby placing increasing influence on observations that are correctly classified with increasing certainty, thereby reducing the relative influence of those incorrectly classified $y_i f(x_i) < 0$.

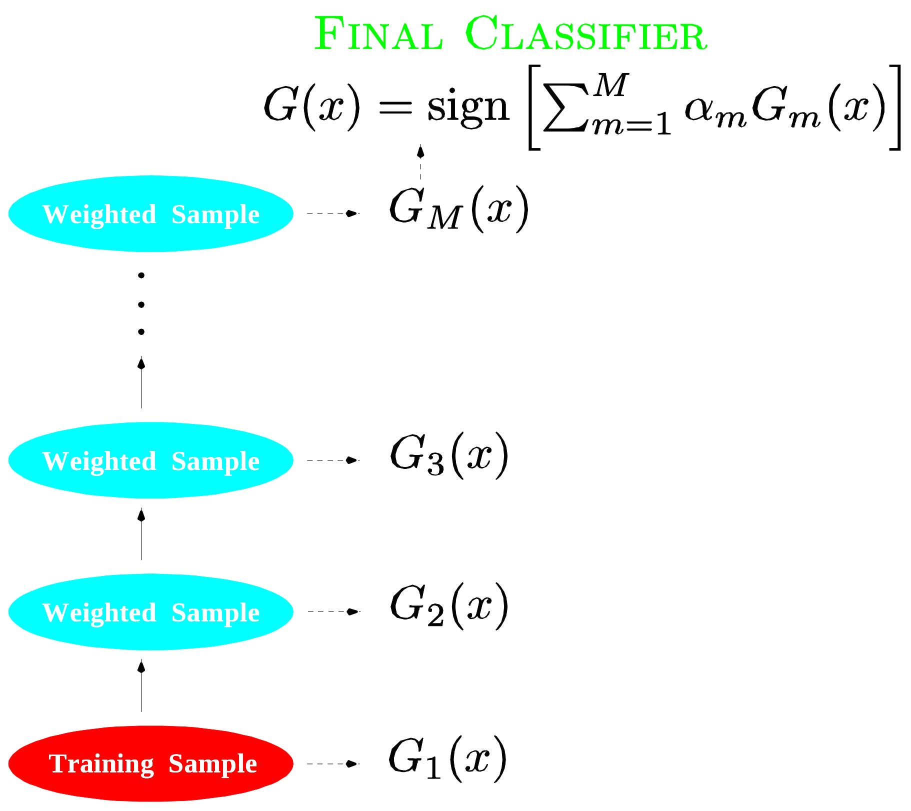

Figure 10.1: *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

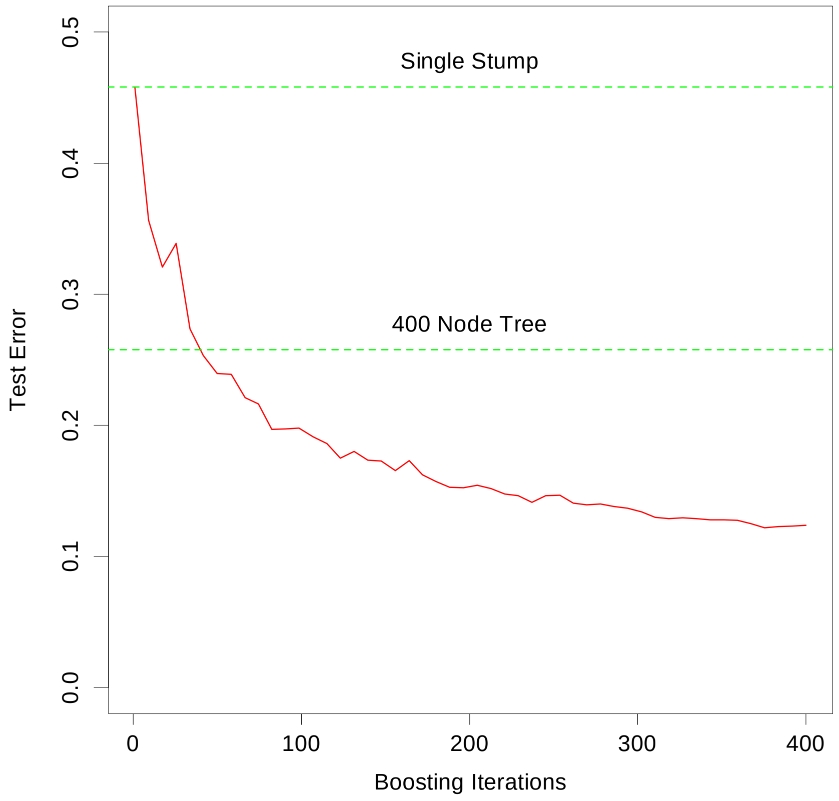

Figure 10.2: *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a* 400 *node classification tree.*

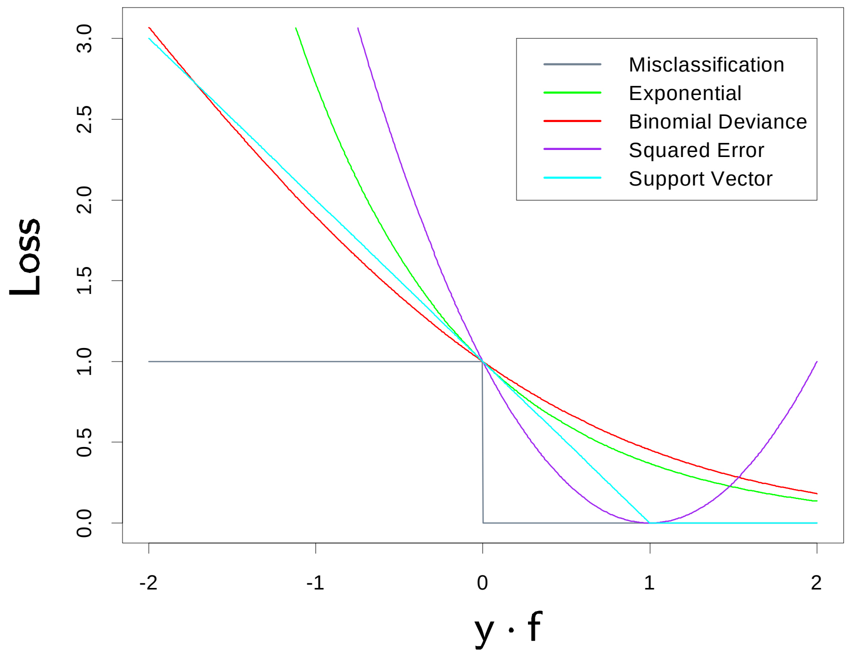

Figure 10.4: *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1+\exp(-2yf))$*; squared error:*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