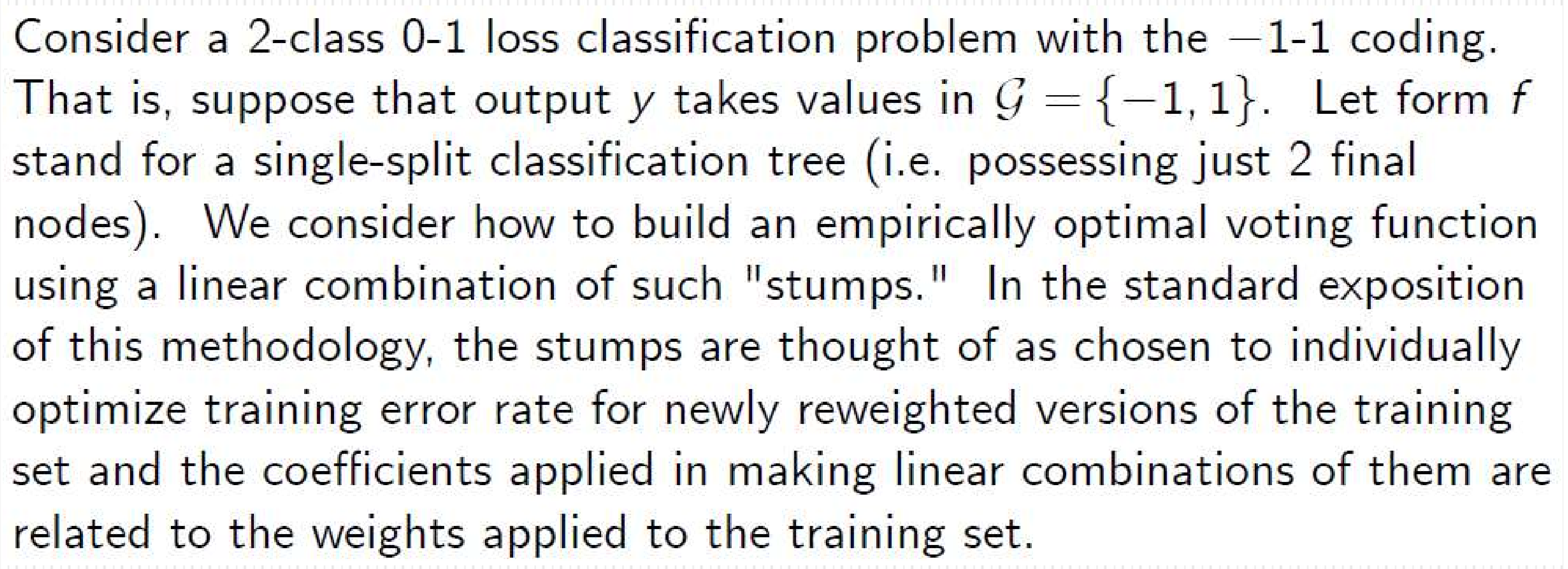
Boosting: AdaBoost.M1

Stephen Vardeman
Analytics Iowa LLC
ISU Statistics and IMSE

AdaBoost.M1 as usually described



Consider a 2-class 0-1 loss classification problem with the -1-1 coding. That is, suppose that output y takes values in $\mathcal{G} = \{-1, 1\}$. Let form f stand for a single-split classification tree (i.e. possessing just 2 final nodes). We consider how to build an empirically optimal voting function using a linear combination of such "stumps." In the standard exposition of this methodology, the stumps are thought of as chosen to individually optimize training error rate for newly reweighted versions of the training set and the coefficients applied in making linear combinations of them are related to the weights applied to the training set.

First iteration

The AdaBoost.M1 algorithm proceeds as follows.

1. Initialize weights on the training data $(\mathbf{x}_i, y_i)$ at

$$w_{i1} \equiv \frac{1}{N} \quad \text{for } i = 1, 2, \ldots, N$$

2. Fit a $\mathcal{G}$-valued "stump" classifier g_1 to the training data to optimize

$$\sum_{i=1}^{N} I\left[y_i \neq g\left(\mathbf{x}_i\right)\right]$$

let

$$\overline{\mathrm{err}}_1 = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq g_1\left(\mathbf{x}_i\right)\right]$$

and define

$$\alpha_1 = \ln\left(\frac{1 - \overline{\mathrm{err}}_1}{\overline{\mathrm{err}}_1}\right)$$

Weight updates and iteration

3. Set new weights on the training data
$$w_{i2} = \frac{1}{N} \exp\left(\alpha_1 I\left[y_i \neq g_1(\mathbf{x}_i)\right]\right) \quad \text{for } i = 1, 2, \ldots, N$$
(This up-weights mis-classified observations by a factor of $\left(1 - \overline{\text{err}}_1\right) / \overline{\text{err}}_1$).)
4. For $m = 2, 3, \ldots, M$
    a. Fit a $\mathcal{G}$-valued predictor/classifier g_m to the training data to optimize
    $$\sum_{i=1}^{N} w_{im} I\left[y_i \neq g(\mathbf{x}_i)\right]$$
    b. Let
    $$\overline{\text{err}}_m = \frac{\sum_{i=1}^{N} w_{im} I\left[y_i \neq g_m(\mathbf{x}_i)\right]}{\sum_{i=1}^{N} w_{im}}$$
    c. and set
    $$\alpha_m = \ln\left(\left(1 - \overline{\text{err}}_m\right) / \overline{\text{err}}_m\right)$$

Iteration and final voting function/classifier

d. Update weights as

$$w_{i(m+1)} = w_{im} \exp\left(\alpha_m I\left[y_i \neq g_m(\mathbf{x}_i)\right]\right)$$
$$= w_{im}\left(I\left[y_i = g_m(\mathbf{x}_i)\right] + \frac{1-\overline{\mathrm{err}}_m}{\overline{\mathrm{err}}_m} I\left[y_i \neq g_m(\mathbf{x}_i)\right]\right)$$

for $i = 1, 2, \ldots, N$. (This up-weights mis-classified observations by a factor of $\left(1-\overline{\mathrm{err}}_m\right)/\overline{\mathrm{err}}_m$).)

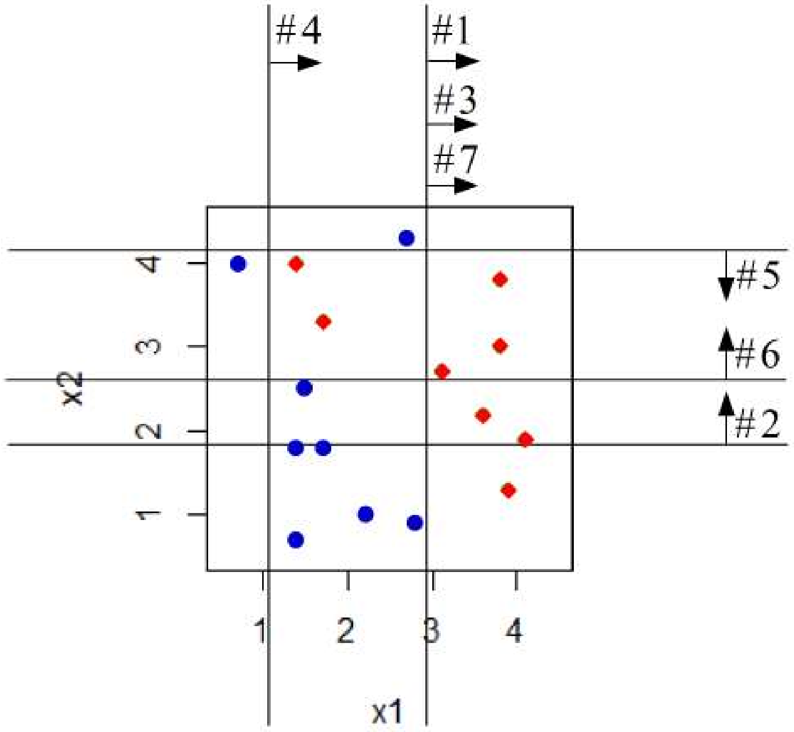
5. Output a resulting classifier that is based on "weighted voting" by the classifiers g_m as

$$\hat{f}_M(\mathbf{x}) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(\mathbf{x})\right)$$

(Classifiers with small $\overline{\mathrm{err}}_m$ get big positive weights in the "voting.")

A toy example

Below is a graphic of a small ($N = 16$) fake $p = 2$ data set and (single line) boundaries of $M = 7$ successive "stumps" used to develop an AdaBoost.M1 classifier with 0 training error rate. (Arrows point in the direction of $y = +1$ decisions.)



Toy example cont'd.

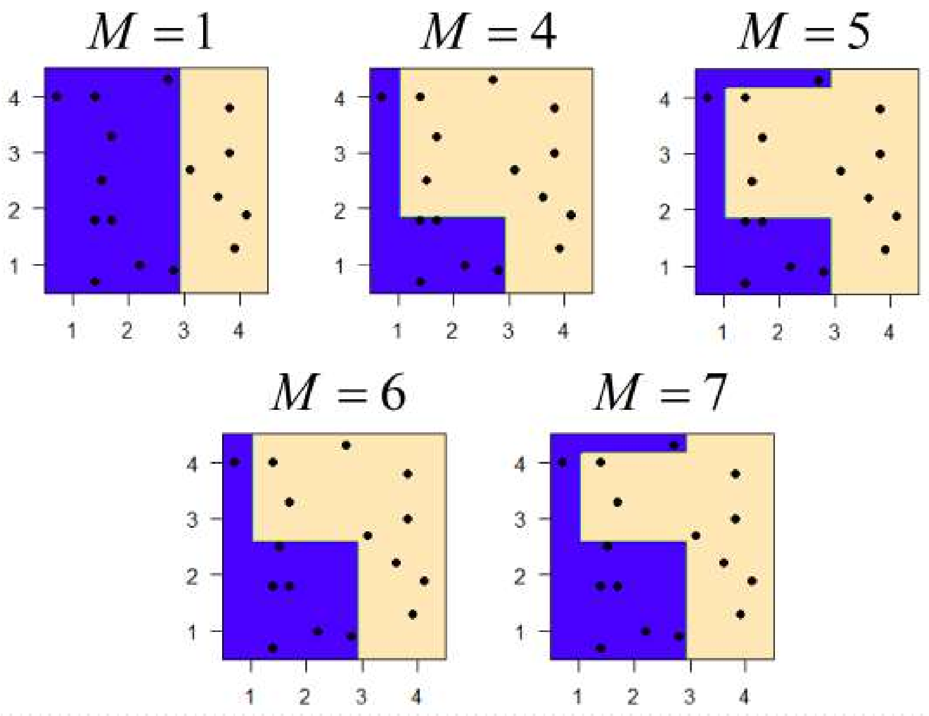
The first cut leaves 2 (red) cases with $y = 1$ on the wrong side of an $M = 1$ decision boundary. The first classifier thus has error rate $\overline{\text{err}}_1 = 2/16$ and $\alpha_1 = \ln((7/8) / (1/8)) = \ln(7)$. The two points on the wrong side of the decision boundary are up-weighted by a factor of 7 in seeking the next stump. One operates as if there are $14 + 7 + 7 = 28$ cases (7 at each of the $\mathbf{x}_i$ that are misclassified at the first cut) in choosing it. This leads to the #2 (horizontal line) decision boundary. And so on. At every iteration m, cases misclassified by the new stump are up-weighted (producing a new total weight). Ultimately, ($-1/1$ valued) stump classifiers $\hat{f}_m$ are combined via

$$\sum_{m=1}^{M} \alpha_m g_m(\mathbf{x})$$

to make the voting function. E.g., the first two terms of this are $\ln(7)\text{sign}(x_1 - 2.95)$ and $\ln(6)\text{sign}(x_2 - 1.85)$.

Toy example cont'd.

Below are graphics portraying the 5 different classifiers met in the development of the 0 training error rate $M = 7$ AdaBoostM.1 classifier.



AdaBoost.M1 and general boosting

There is an argument in Section 11.4.4 of the notes that shows that, in fact, the AdaBoost.M1 algorithm is a general boosting algorithm for development of a voting function based on the exponential loss and choice of exactly-optimal-stump-updates of iterates. It is thus not surprising that AdaBoost.M1 has been called a "best existing off-the-shelf technology" for 2-class classification. Of course it is. It is adequately flexible to approximate an optimal voting function (half a likelihood ratio) to the precision provided by the training data, and its successive approximation character pushes iterates in exactly that direction.