# 2

# *Boosting*

## 2.1 A General Boosting Procedure

The term **boosting** refers to a family of algorithms that are able to convert weak learners to strong learners. Intuitively, a weak learner is just slightly better than random guess, while a strong learner is very close to perfect performance. The birth of boosting algorithms was originated from the answer to an interesting theoretical question posed by Kearns and Valiant [1989]. That is, whether two complexity classes, *weakly learnable* and *strongly learnable* problems, are equal. This question is of fundamental importance, since if the answer is positive, any weak learner is potentially able to be boosted to a strong learner; particularly if we note that in real practice it is generally very easy to obtain weak learners but difficult to get strong learners. Schapire [1990] proved that the answer is positive, and the proof is a construction, i.e., boosting.

The general boosting procedure is quite simple. Suppose the weak learners will work on any data distribution it is given, and take the binary classification task as an example; that is, we are trying to classify instances as *positive* and *negative*. The training instances in space $\mathcal{X}$ are drawn *i.i.d.* from distribution $\mathcal{D}$, and the ground-truth function is $f$. Suppose the space $\mathcal{X}$ is composed of three parts $\mathcal{X}_1, \mathcal{X}_2$ and $\mathcal{X}_3$, each takes $1/3$ amount of the distribution, and a learner working by random guess has $50\%$ classification error on this problem. We want to get an accurate (e.g., zero error) classifier on the problem, but we are unlucky and only have a weak classifier at hand, which only has correct classifications in spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ and has wrong classifications in $\mathcal{X}_3$, thus has $1/3$ classification error. Let's denote this weak classifier as $h_1$. It is obvious that $h_1$ is not desired.

The idea of boosting is to correct the mistakes made by $h_1$. We can try to derive a new distribution $\mathcal{D}'$ from $\mathcal{D}$, which makes the mistakes of $h_1$ more evident, e.g., it focuses more on the instances in $\mathcal{X}_3$. Then, we can train a classifier $h_2$ from $\mathcal{D}'$. Again, suppose we are unlucky and $h_2$ is also a weak classifier, which has correct classifications in $\mathcal{X}_1$ and $\mathcal{X}_3$ and has wrong classifications in $\mathcal{X}_2$. By combining $h_1$ and $h_2$ in an appropriate way (we will explain how to combine them in the next section), the combined classifier will have correct classifications on $\mathcal{X}_1$, and maybe some errors in $\mathcal{X}_2$ and $\mathcal{X}_3$.

```
Input: Sample distribution $\mathcal{D}$;
       Base learning algorithm $\mathfrak{L}$;
       Number of learning rounds $T$.
Process:
    $\mathcal{D}_1 = \mathcal{D}$.    % Initialize distribution
    for $t = 1, \ldots, T$:
        $h_t = \mathfrak{L}(\mathcal{D}_t)$;    % Train a weak learner from distribution $\mathcal{D}_t$
        $\epsilon_t = P_{\boldsymbol{x} \sim D_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;    % Evaluate the error of $h_t$
        $\mathcal{D}_{t+1} = Adjust\_Distribution(\mathcal{D}_t, \epsilon_t)$
    end
Output: $H(\boldsymbol{x}) = Combine\_Outputs(\{h_t(\boldsymbol{x})\})$
```

FIGURE 2.1: A general boosting procedure

Again, we derive a new distribution $\mathcal{D}''$ to make the mistakes of the combined classifier more evident, and train a classifier $h_3$ from the distribution, so that $h_3$ has correct classifications in $\mathcal{X}_2$ and $\mathcal{X}_3$. Then, by combining $h_1$, $h_2$ and $h_3$, we have a perfect classifier, since in each spaces of $\mathcal{X}_1$, $\mathcal{X}_2$ and $\mathcal{X}_3$, at least two classifiers make correct classifications.

Briefly, boosting works by training a set of learners sequentially and combining them for prediction, where the later learners focus more on the mistakes of the earlier learners. Figure 2.1 summarizes the general boosting procedure.

## 2.2 The AdaBoost Algorithm

The general boosting procedure described in Figure 2.1 is not a real algorithm since there are some unspecified parts such as $Adjust\_Distribution$ and $Combine\_Outputs$. The AdaBoost algorithm [Freund and Schapire, 1997], which is the most influential boosting algorithm, can be viewed as an instantiation of these parts as shown in Figure 2.2.

One version of derivation of AdaBoost [Friedman et al., 2000] is achieved by minimizing the **exponential loss** function

$$\ell_{\exp}(h \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}}[e^{-f(\boldsymbol{x})h(\boldsymbol{x})}] \tag{2.1}$$

using *additive* weighted combination of weak learners as

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \ . \tag{2.2}$$

```
**Input:** Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
           Base learning algorithm $\mathfrak{L}$;
           Number of learning rounds $T$.
**Process:**
$\mathcal{D}_1(\boldsymbol{x}) = 1/m$.   % Initialize the weight distribution
**for** $t = 1, \ldots, T$:
    $h_t = \mathfrak{L}(D, \mathcal{D}_t)$;  % Train a classifier $h_t$ from $D$ under distribution $\mathcal{D}_t$
    $\epsilon_t = P_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;  % Evaluate the error of $h_t$
    **if** $\epsilon_t > 0.5$ **then break**
    $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$;  % Determine the weight of $h_t$
    $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases}$
        $= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$  % Update the distribution, where
                                          % $Z_t$ is a normalization factor which
                                          % enables $\mathcal{D}_{t+1}$ to be a distribution
**end**
**Output:** $H(\boldsymbol{x}) = \mathtt{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$
```

FIGURE 2.2: The AdaBoost algorithm

The exponential loss is used since it gives an elegant and simple update formula, and it is consistent with the goal of minimizing classification error and can be justified by its relationship to the standard log likelihood. When the exponential loss is minimized by $H$, the partial derivative of the exponential loss for every $\boldsymbol{x}$ is zero, i.e.,

$$\begin{aligned} \frac{\partial e^{-f(\boldsymbol{x})H(\boldsymbol{x})}}{\partial H(\boldsymbol{x})} &= -f(\boldsymbol{x}) e^{-f(\boldsymbol{x})H(\boldsymbol{x})} \\ &= -e^{-H(\boldsymbol{x})} P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) + e^{H(\boldsymbol{x})} P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) \\ &= 0\,. \end{aligned} \tag{2.3}$$

Then, by solving (2.3), we have

$$H(\boldsymbol{x}) = \frac{1}{2} \ln \frac{P(f(x) = 1 \mid \boldsymbol{x})}{P(f(x) = -1 \mid \boldsymbol{x})}\,, \tag{2.4}$$

and hence,

$$\begin{aligned}\texttt{sign}(H(\boldsymbol{x})) &= \texttt{sign}\left(\frac{1}{2}\ln\frac{P(f(x)=1\mid\boldsymbol{x})}{P(f(x)=-1\mid\boldsymbol{x})}\right)\\ &= \begin{cases}1, & P(f(x)=1\mid\boldsymbol{x}) > P(f(x)=-1\mid\boldsymbol{x})\\ -1, & P(f(x)=1\mid\boldsymbol{x}) < P(f(x)=-1\mid\boldsymbol{x})\end{cases}\\ &= \mathop{\arg\max}_{y\in\{-1,1\}} P(f(x)=y\mid\boldsymbol{x}),\end{aligned} \tag{2.5}$$

which implies that $\texttt{sign}(H(\boldsymbol{x}))$ achieves the Bayes error rate. Note that we ignore the case $P(f(x)=1\mid\boldsymbol{x}) = P(f(x)=-1\mid\boldsymbol{x})$. The above derivation shows that when the exponential loss is minimized, the classification error is also minimized, and thus the exponential loss is a proper optimization target for replacing the non-differentiable classification error.

The $H$ is produced by iteratively generating $h_t$ and $\alpha_t$. The first weak classifier $h_1$ is generated by invoking the weak learning algorithm on the original distribution. When a classifier $h_t$ is generated under the distribution $\mathcal{D}_t$, its weight $\alpha_t$ is to be determined such that $\alpha_t h_t$ minimizes the exponential loss

$$\begin{aligned}\ell_{\exp}(\alpha_t h_t \mid \mathcal{D}_t) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}[e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}] \qquad (2.6)\\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-\alpha_t}\mathbb{I}(f(\boldsymbol{x})=h_t(\boldsymbol{x})) + e^{\alpha_t}\mathbb{I}(f(\boldsymbol{x})\neq h_t(\boldsymbol{x}))\right]\\ &= e^{-\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x})=h_t(\boldsymbol{x})) + e^{\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x})\neq h_t(\boldsymbol{x}))\\ &= e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t\ ,\end{aligned}$$

where $\epsilon_t = P_{\boldsymbol{x}\sim\mathcal{D}_t}(h_t(\boldsymbol{x})\neq f(\boldsymbol{x}))$. To get the optimal $\alpha_t$, let the derivative of the exponential loss equal zero, that is,

$$\frac{\partial\ell_{\exp}(\alpha_t h_t\mid\mathcal{D}_t)}{\partial\alpha_t} = -e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t = 0\ , \tag{2.7}$$

then the solution is

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\ , \tag{2.8}$$

as in the line 6 of Figure 2.2.

Once a sequence of weak classifiers and their corresponding weights have been generated, these classifiers are combined as $H_{t-1}$. Then, AdaBoost adjusts the sample distribution such that in the next round, the base learning algorithm will output a weak classifier $h_t$ that corrects some mistakes of $H_{t-1}$. Considering the exponential loss again, the ideal classifier $h_t$ that corrects all mistakes of $H_{t-1}$ should minimize the exponential loss

$$\begin{aligned}\ell_{\exp}(H_{t-1}+h_t\mid\mathcal{D}) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x}))}] \qquad (2.9)\\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}]\ .\end{aligned}$$

Using Taylor expansion of $e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}$, the exponential loss is approximated by

$$\begin{aligned}\ell_{\exp}(H_{t-1}+h_t \mid \mathcal{D}) &\approx \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{f(\boldsymbol{x})^2h_t(\boldsymbol{x})^2}{2}\right)\right]\\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{1}{2}\right)\right]\ , \qquad (2.10)\end{aligned}$$

by noticing that $f(\boldsymbol{x})^2=1$ and $h_t(\boldsymbol{x})^2=1$.

Thus, the ideal classifier $h_t$ is

$$\begin{aligned}h_t(\boldsymbol{x}) &= \arg\min_h \ell_{\exp}(H_{t-1}+h \mid \mathcal{D}) \qquad (2.11)\\ &= \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h(\boldsymbol{x})+\frac{1}{2}\right)\right]\\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}f(\boldsymbol{x})h(\boldsymbol{x})\right]\\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]}f(\boldsymbol{x})h(\boldsymbol{x})\right],\end{aligned}$$

by noticing that $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]$ is a constant.

Denote a distribution $\mathcal{D}_t$ as

$$\mathcal{D}_t(x) = \frac{\mathcal{D}(x)e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]}\ . \qquad (2.12)$$

Then, by the definition of mathematical expectation, it is equivalent to write that

$$\begin{aligned}h_t(\boldsymbol{x}) &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]}f(\boldsymbol{x})h(\boldsymbol{x})\right] \qquad (2.13)\\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[f(\boldsymbol{x})h(\boldsymbol{x})\right]\ .\end{aligned}$$

Further noticing that $f(\boldsymbol{x})h_t(\boldsymbol{x}) = 1-2\mathbb{I}(f(\boldsymbol{x})\neq h_t(\boldsymbol{x}))$, the ideal classifier is

$$h_t(\boldsymbol{x}) = \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[\mathbb{I}(f(\boldsymbol{x})\neq h(\boldsymbol{x}))\right]\ . \qquad (2.14)$$

As can be seen, the ideal $h_t$ minimizes the classification error under the distribution $\mathcal{D}_t$. Therefore, the weak learner is to be trained under $\mathcal{D}_t$, and has less than 0.5 classification error according to $\mathcal{D}_t$. Considering the relation between $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$, we have

$$\begin{aligned}\mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{\mathcal{D}(x)e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}]} \\ &= \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}]} \\ &= \mathcal{D}_t(\boldsymbol{x}) \cdot e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})} \frac{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}]} \ ,\end{aligned} \tag{2.15}$$

which is the way AdaBoost updates the sample distribution as in the line 7 of Figure 2.2.

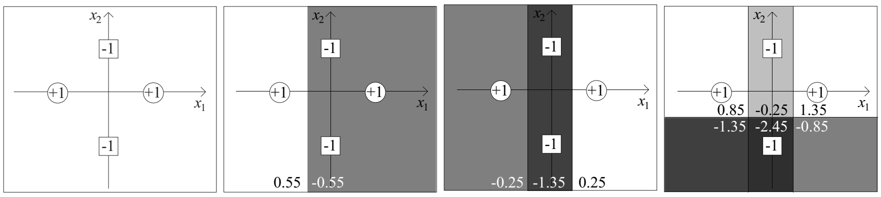
It is noteworthy that the AdaBoost algorithm described in Figure 2.2 requires the base learning algorithm being able to learn with specified distributions. This is often accomplished by **re-weighting**, that is, weighting training examples in each round according to the sample distribution. For base learning algorithms that cannot handle weighted training examples, **re-sampling**, that is, sampling training examples in each round according to the desired distribution, can be applied.

For base learning algorithms which can be used with both re-weighting and re-sampling, generally there is no clear performance difference between these two implementations. However, re-sampling provides an option for **Boosting with restart** [Kohavi and Wolpert, 1996]. In each round of AdaBoost, there is a *sanity check* to ensure that the current base learner is better than random guess (see the the line 5 of Figure 2.2). This sanity check might be violated on some tasks when there are only a few weak learners and the AdaBoost procedure will be early-terminated far before the specified number of rounds $T$. This occurs particularly often on multiclass tasks. When re-sampling is used, the base learner that cannot pass the sanity check can be removed, and a new data sample can be generated, on which a new base learner will be trained; in this way, the AdaBoost procedure can avoid the early-termination problem.

## 2.3 Illustrative Examples

It is helpful to gain intuitive understanding of AdaBoost by observing its behavior. Consider an artificial data set in a two-dimensional space, plotted in Figure 2.3(a). There are only four instances, i.e.,

$$\left\{\begin{array}{l}(\boldsymbol{z}_1 = (+1,0)\,, y_1 = +1) \\ (\boldsymbol{z}_2 = (-1,0)\,, y_2 = +1) \\ (\boldsymbol{z}_3 = (0,+1)\,, y_3 = -1) \\ (\boldsymbol{z}_4 = (0,-1)\,, y_4 = -1)\end{array}\right\} ,$$



(a) The XOR data (b) 1st round (c) 2nd round (d) 3rd round

FIGURE 2.3: AdaBoost on the XOR problem.

where $y_i = f(\boldsymbol{z}_i)$ is the label of each instance. This is the XOR problem. As can be seen, there is no straight line that is able to separate positive instances (i.e., $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$) from negative instances (i.e., $\boldsymbol{z}_3$ and $\boldsymbol{z}_4$); in other words, the two classes cannot be separated by a linear classifier.

Suppose we have a base learning algorithm which works as follows. It evaluates eight basis functions $h_1$ to $h_8$ described in Figure 2.4 on the training data under a given distribution, and then outputs the one with the smallest error. If there is more than one basis function with the smallest error, it selects one randomly. Notice that none of these eight basis functions can separate the two classes.

Now we track how AdaBoost works:

1. The first step is to invoke the base learning algorithm on the original data. Since $h_2$, $h_3$, $h_5$ and $h_8$ all have the smallest classification errors 0.25, suppose the base learning algorithm outputs $h_2$ as the classifier. After that, one instance, $\boldsymbol{z}_1$, is incorrectly classified, so the error is

$$h_1(\boldsymbol{x}) = \begin{cases} +1, \text{ if } (x_1 > -0.5) \\ -1, \quad \text{otherwise} \end{cases} \quad h_2(\boldsymbol{x}) = \begin{cases} -1, \text{ if } (x_1 > -0.5) \\ +1, \quad \text{otherwise} \end{cases}$$

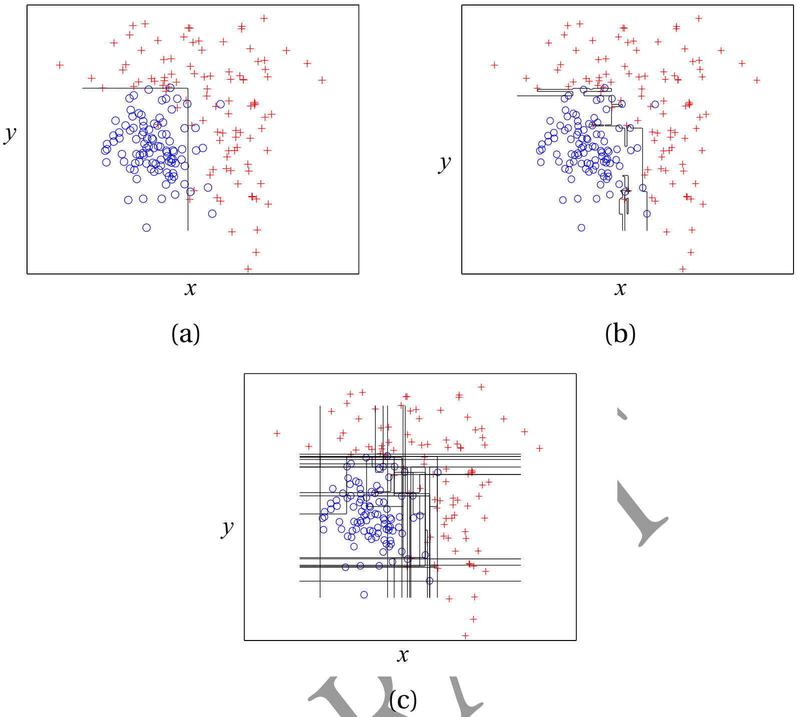
$$h_3(\boldsymbol{x}) = \begin{cases} +1, \text{ if } (x_1 > +0.5) \\ -1, \quad \text{otherwise} \end{cases} \quad h_4(\boldsymbol{x}) = \begin{cases} -1, \text{ if } (x_1 > +0.5) \\ +1, \quad \text{otherwise} \end{cases}$$

$$h_5(\boldsymbol{x}) = \begin{cases} +1, \text{ if } (x_2 > -0.5) \\ -1, \quad \text{otherwise} \end{cases} \quad h_6(\boldsymbol{x}) = \begin{cases} -1, \text{ if } (x_2 > -0.5) \\ +1, \quad \text{otherwise} \end{cases}$$

$$h_7(\boldsymbol{x}) = \begin{cases} +1, \text{ if } (x_2 > +0.5) \\ -1, \quad \text{otherwise} \end{cases} \quad h_8(\boldsymbol{x}) = \begin{cases} -1, \text{ if } (x_2 > +0.5) \\ +1, \quad \text{otherwise} \end{cases}$$

where $x_1$ and $x_2$ are the values of $\boldsymbol{x}$ at the first and the second dimension, respectively.

FIGURE 2.4: The eight basis functions considered by the base learning algorithm.



(a)

(b)

(c)

FIGURE 2.5: Decision boundaries of (a) a single decision tree, (b) AdaBoost and (c) the 10 decision trees used by AdaBoost, on the *three-gaussians* data set.

$1/4 = 0.25$. The weight of $h_2$ is $0.5 \ln 3 \approx 0.55$. Figure 2.3(b) visualizes the classification, where the shadow area is classified as negative (-1) and the weights of the classification, 0.55 and -0.55, are displayed.

2. The weight of $z_1$ is increased, and the base learning algorithm is invoked again. This time $h_3$, $h_5$ and $h_8$ have the smallest error, and suppose $h_3$ is picked, of which the weight is 0.80. Figure 2.3(c) shows the combined classification of $h_2$ and $h_3$ with their weights, where different gray levels are used for distinguishing the negative areas according to the combination weights.
3. The weight of $z_2$ is increased. This time only $h_5$ and $h_8$ equally have the smallest errors, and suppose $h_5$ is picked, of which the weight is 1.10. Figure 2.3(d) shows the combined classification of $h_2$, $h_3$ and $h_5$.

After the three steps, let us consider the sign of classification weights in each area in Figure 2.3(d). It can be observed that the sign of classification weights of $z_1$ and $z_2$ is '+', while that of $z_3$ and $z_4$ is '$-$'. This means all the

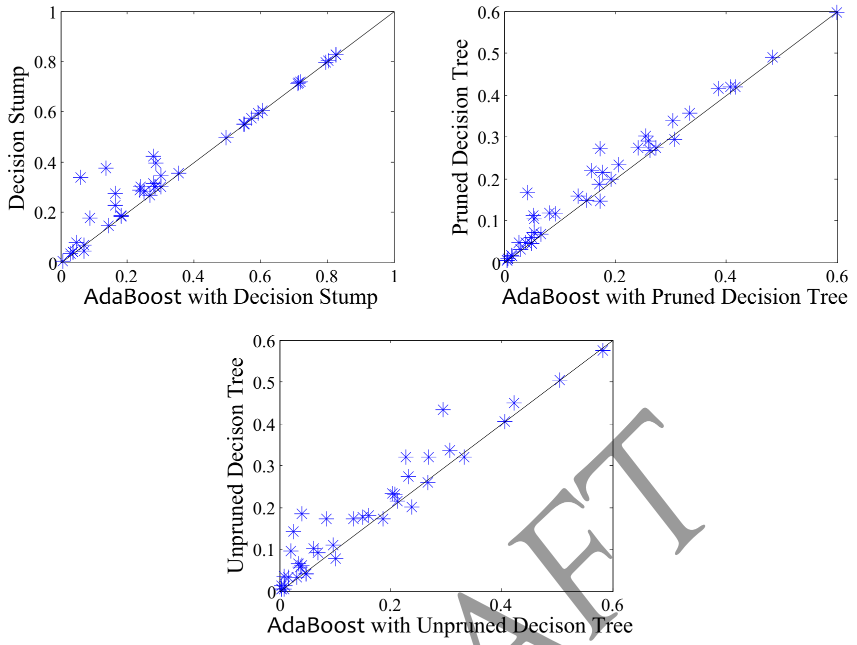


FIGURE 2.6: Comparison of predictive errors of AdaBoost against single base learners on 40 UCI data sets. Each point represents a data set and locates according to the predictive error of the two compared algorithms. The diagonal line indicates where the two compared algorithms have identical errors.

instances are correctly classified; thus, by combining the imperfect linear classifiers, AdaBoost has produced a non-linear classifier with zero error.

For a further understanding of AdaBoost, we visualize the decision boundaries of a single decision tree, AdaBoost and its component decision trees on the *three-guassians* data set, as shown in Figure 2.5. It can be observed that the decision boundary of AdaBoost is more flexible than that of a single decision tree, and this helps to reduce the error from 9.4% of the single decision tree to 8.3% of the boosted ensemble.

We also evaluate the AdaBoost algorithm on 40 data sets from the UCI Machine Learning Repository [1], which covers a broad range of real-world tasks. The Weka [2] implementation of AdaBoost.M1 using re-weighting with 50 weak learners is evaluated. Almost all kinds of learning algorithms can

[1] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[2] `http://www.cs.waikato.ac.nz/ml/weka/`

be taken as base learning algorithms, such as decision trees, neural networks, etc. Here, we have tried three base learning algorithms: decision stumps, and pruned and unpruned J48 decision trees (Weka implementation of C4.5 decision trees). We plot the comparison results in Figure 2.6, from which it can be observed that AdaBoost usually outperforms its base learning algorithm, with only a few exceptions on which it hurts performance.

## 2.4 Theoretical Issues

### 2.4.1 Initial Analysis

Freund and Schapire [1997] proved that, if the base learners of AdaBoost have errors $\epsilon_1$, $\epsilon_2$, ..., $\epsilon_T$, the error of the final combined learner, $\epsilon$, is upper bounded by

$$\epsilon = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\mathbb{I}[H(\boldsymbol{x}) \neq f(\boldsymbol{x})] \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \leq e^{-2\sum_{t=1}^{T}\gamma_t^2} \ , \qquad (2.16)$$

where $\gamma_t = 0.5 - \epsilon_t$ is called the **edge** of $h_t$. It can be seen that AdaBoost reduces the error exponentially fast. Also, it can be derived that, to achieve an error less than $\epsilon$, the number of learning rounds $T$ is upper bounded by

$$T \leq \lceil \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} \rceil \ , \qquad (2.17)$$

where it is assumed that $\gamma \geq \gamma_1 \geq \ldots \geq \gamma_T$.

In practice, however, all the estimates can only be carried out on training data $D$, i.e., $\epsilon_D = \mathbb{E}_{\boldsymbol{x}\sim D}\mathbb{I}[H(\boldsymbol{x}) \neq f(\boldsymbol{x})]$, and thus the errors are training errors, while the generalization error $\epsilon_{\mathcal{D}}$ is of more interest. Freund and Schapire [1997] showed that the generalization error of AdaBoost is upper bounded by

$$\epsilon_{\mathcal{D}} \leq \epsilon_D + \tilde{O}\left(\sqrt{\frac{dT}{m}}\right) \qquad (2.18)$$

with probability at least $1-\delta$, where $d$ is the **VC-dimension** of base learners, $m$ is the number of training instances, $T$ is the number of learning rounds, and $\tilde{O}(\cdot)$ is used instead of $O(\cdot)$ to hide logarithmic terms and constant factors.

### 2.4.2 Margin Explanation

The generalization bound (2.18) suggests that, in order to achieve a good generalization, it is necessary to constrain the complexity of base learners

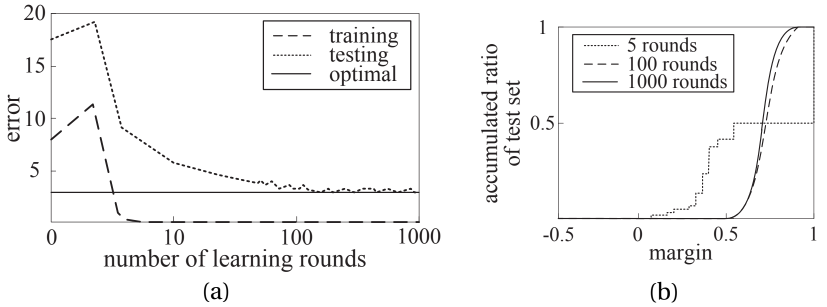


FIGURE 2.7: (a) Training and test error, and (b) margin distribution of AdaBoost on the UCI *letter* data set. (Plot based on a similar figure in [Schapire et al., 1998])

as well as the number of learning rounds, and otherwise AdaBoost will overfit.

Empirical studies, however, show that AdaBoost often does not overfit; that is, the test error often tends to decrease even after the training error reaches zero, even after a large number of rounds such as 1,000. For example, Schapire et al. [1998] plotted the typical performance of AdaBoost as shown in Figure 2.7(a). It can be observed that AdaBoost achieves zero training error in less than 10 rounds but after that, the generalization error keeps reducing. This phenomenon seems contradicting with the Occam's Razor which prefers simple hypothesis to complex ones when both fit empirical observations well. So, it is not strange that explaining why AdaBoost seems resistant to overfitting becomes one of the central theoretical issues and has attracted much attention.

Schapire et al. [1998] introduced the margin-based explanation to AdaBoost. Formally, in the context of binary classification, i.e., $f(\boldsymbol{x}) \in \{-1, +1\}$, the **margin** of the classifier $h$ on the instance $\boldsymbol{x}$, or in other words, the distance of $\boldsymbol{x}$ to the classification hyperplane of $h$, is defined as $f(\boldsymbol{x})h(\boldsymbol{x})$, and similarly, the margin of the ensemble $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$ is $f(\boldsymbol{x})H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t f(\boldsymbol{x})h_t(\boldsymbol{x})$, while the **normalized margin** of the ensemble is

$$f(\boldsymbol{x})H(\boldsymbol{x}) = \frac{\sum_{t=1}^{T} \alpha_t f(\boldsymbol{x})h_t(\boldsymbol{x})}{\sum_{t=1}^{T} \alpha_t} , \tag{2.19}$$

where $\alpha_t$'s are the weights of base learners.

Based on the concept of margin, Schapire et al. [1998] proved that, given any threshold $\theta > 0$ of margin over the training sample $D$, with probability at least $1 - \delta$, the generalization error of the ensemble $\epsilon_{\mathcal{D}} = P_{\boldsymbol{x}\sim\mathcal{D}}(f(\boldsymbol{x}) \neq$

$H(\boldsymbol{x})$) can be bounded by

$$\begin{aligned}\epsilon_{\mathcal{D}} &\leq P_{\boldsymbol{x}\sim D}(f(\boldsymbol{x})H(\boldsymbol{x}) \leq \theta) + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2} + \ln\frac{1}{\delta}}\right) \\ &\leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2} + \ln\frac{1}{\delta}}\right) ,\end{aligned} \tag{2.20}$$

where $d$, $m$, $T$ and $\tilde{O}(\cdot)$ are as same as those in (2.18), and $\epsilon_t$ is the training error of the base learner $h_t$. The bound (2.20) discloses that when other variables are fixed, the larger the margin over the training set, the smaller the generalization error. Thus, Schapire et al. [1998] argued that AdaBoost tends to be resistant to overfitting since it is able to increase the ensemble margin even after the training error reaches zero. Figure 2.7(b) illustrates the margin distribution of AdaBoost at different number of learning rounds.

Notice that the bound (2.20) depends heavily on the smallest margin, since the probability $P_{\boldsymbol{x}\sim D}(f(\boldsymbol{x})H(\boldsymbol{x}) \leq \theta)$ will be small if the smallest margin is large. Based on this recognition, Breiman [1999] developed the arc-gv algorithm, which is a variant of AdaBoost but directly maximizes the **minimum margin**

$$\varrho = \min_{\boldsymbol{x}\in D} f(\boldsymbol{x})H(\boldsymbol{x}) . \tag{2.21}$$

In each round, arc-gv updates $\alpha_t$ according to

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1+\gamma_t}{1-\gamma_t}\right) - \frac{1}{2}\ln\left(\frac{1+\varrho_t}{1-\varrho_t}\right) , \tag{2.22}$$

where $\gamma_t$ is the edge of $h_t$, and $\varrho_t$ is the minimum margin of the combined classifier up to the current round.

Based on the minimum margin, Breiman [1999] proved a generalization error bound tighter than (2.20). Since the minimum margin of arc-gv converges to the largest possible minimum margin, the margin theory would appear to predict that arc-gv should perform better than AdaBoost. However, Breiman [1999] found in experiments that, though arc-gv does produce uniformly larger minimum margin than AdaBoost, the test error of arc-gv increases drastically in almost every case. Hence, Breiman [1999] convincingly concluded that the margin-based explanation for AdaBoost was in serious doubt and a new understanding is needed. This almost sentenced the margin theory to death.

Seven years later, Reyzin and Schapire [2006] reported an interesting finding. The bound of generalization error (2.20) is relevant to the margin, the number of learning rounds and the complexity of base learners. To study the influence of margin, the other factors should be fixed. When comparing arc-gv and AdaBoost, Breiman [1999] tried to control the complexity of

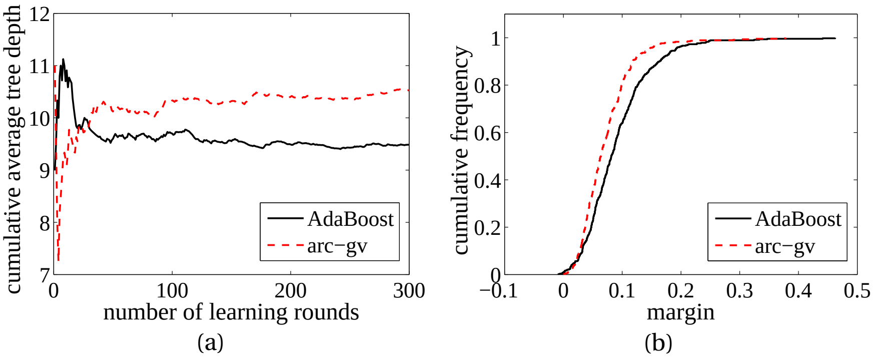


FIGURE 2.8: (a) Tree depth and (b) margin distribution of AdaBoost against arc-gv on the UCI *clean1* data set.

base learners by using decision trees with a fixed number of leaves. However, Reyzin and Schapire [2006] found that these trees have very different shapes. The trees generated by arc-gv tend to have larger depth, while those generated by AdaBoost tend to have larger width. Figure 2.8(a) depicts the difference of the depth of typical trees generated by the two algorithms. Though the trees have the same number of leaves, it seems that a deeper tree makes more attribute tests than a wider tree, and therefore they are unlikely to have equal complexity. So, Reyzin and Schapire [2006] repeated Breiman's experiments by using decision stumps which have only two leaves and therefore have a fixed complexity, and found that the **margin distribution** of AdaBoost is better than that of arc-gv, as illustrated in Figure 2.8(b).

Thus, the margin distribution is believed crucial to the generalization performance of AdaBoost, and Reyzin and Schapire [2006] suggested to consider *average margin* or *median margin* as measures to compare margin distributions.

### 2.4.3 Statistical View

Though the margin-based explanation to AdaBoost has nice geometrical intuition and is attractive to the learning community, it is not that attractive to the statistics community, and statisticians have tried to understand AdaBoost from the perspective of statistical methods. A breakthrough in this direction was made by Friedman et al. [2000] who showed that the AdaBoost algorithm can be interpreted as a stagewise estimation procedure for fitting an additive logistic regression model, which is exactly how we derive the AdaBoost in Section 2.2.

Noticing that (2.2) is a form of **additive model**. The exponential loss

```
Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
       Least square base learning algorithm $\mathfrak{L}$;
       Number of learning rounds $T$.
Process:
$y_0(\boldsymbol{x}) = f(\boldsymbol{x})$.    % Initialize target
$H_0(\boldsymbol{x}) = 0$.    % Initialize function
for $t = 1, \ldots, T$:
    $p_t(\boldsymbol{x}) = \frac{1}{1+e^{-2H_{t-1}(\boldsymbol{x})}}$;  %Calculate probability
    $y_t(\boldsymbol{x}) = \frac{y_{t-1}(\boldsymbol{x}) - p_{t-1}(\boldsymbol{x})}{p_{t-1}(\boldsymbol{x})(1-p_{t-1}(\boldsymbol{x}))}$; % Update target
    $\mathcal{D}_t(\boldsymbol{x}) = p_{t-1}(\boldsymbol{x})(1 - p_{t-1}(\boldsymbol{x}))$; % Update weight
    $h_t = \mathfrak{L}(D, y_t, \mathcal{D}_t)$;  % Train a least square classifier $h_t$ to fit $y_t$
                                 in data set $D$ under distribution $\mathcal{D}_t$
    $H_t(\boldsymbol{x}) = H_{t-1}(\boldsymbol{x}) + \frac{1}{2}h_t(\boldsymbol{x})$;  %Update combined classifier
end
Output: $H(\boldsymbol{x}) = \mathtt{sign}\left(\sum_{t=1}^{T} h_t(\boldsymbol{x})\right)$
```

FIGURE 2.9: The LogitBoost algorithm

function (2.1) adopted by AdaBoost is a differentiable upper bound of the 0/1-loss function that is typically used for measuring misclassification error [Schapire and Singer, 1999]. If we take a logistic function and estimate probability via

$$P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) = \frac{e^{H(\boldsymbol{x})}}{e^{H(\boldsymbol{x})} + e^{-H(\boldsymbol{x})}} , \tag{2.23}$$

we can find that the exponential loss function and the *log loss* function (negative log-likelihood)

$$\ell_{log}(h \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}} \left[ \ln \left( 1 + e^{-2f(\boldsymbol{x})h(\boldsymbol{x})} \right) \right] \tag{2.24}$$

are minimized by the same function (2.4). So, instead of taking the Newton-like updates in AdaBoost, Friedman et al. [2000] suggested to fit the additive logistic regression model by optimizing the log loss function via gradient decent with the base regression models, leading to the LogitBoost algorithm shown in Figure 2.9.

According to the explanation of Friedman et al. [2000], AdaBoost is just an optimization process that tries to fit an additive model based on a **surrogate loss function**. Ideally, a surrogate loss function should be *consistent*, i.e., optimizing the surrogate loss will yield ultimately an optimal function with the Bayes error rate for true loss function, while the optimization of the surrogate loss is computationally more efficient. Many variants of AdaBoost have been developed by considering different surrogate loss functions, e.g., the LogitBoost which considers the log loss, the L2Boost which considers the $l_2$ *loss* [Bühlmann and Yu, 2003], etc.

On the other hand, if we just regard a boosting procedure as an optimization of a loss function, an alternative way for this purpose is to use mathematical programming [Demiriz et al., 2002, Warmuth et al., 2008] to solve the weights of weak learners. Consider an additive model $\sum_{h \in \mathcal{H}} \alpha_h h$ of a pool $\mathcal{H}$ of weak learners, and let $\xi_i$ be the loss of the model on instance $\boldsymbol{x}_i$. Demiriz et al. [2002] derived that, if the sum of coefficients and losses is bounded such that

$$\sum_{h \in \mathcal{H}} \alpha_h + C \sum_{i=1}^{m} \xi_i \le B \ , \tag{2.25}$$

which actually bounds the complexity (or **covering number**) of the model [Zhang, 1999], the generalization error is therefore bounded as

$$\epsilon_{\mathfrak{D}} \le \tilde{O}\left(\frac{\ln m}{m} B^2 \ln(Bm) + \frac{1}{m} \ln \frac{1}{\delta}\right) \ , \tag{2.26}$$

where $C \ge 1$ and $\alpha_h \ge 0$, and $\tilde{O}$ hides other variables. It is evident that minimizing $B$ also minimizes this upper bound. Thus, considering $T$ weak learners, letting $y_i = f(\boldsymbol{x}_i)$ be the label of training instance $\boldsymbol{x}_i$ and $H_{i,j} = h_j(\boldsymbol{x}_i)$ be the output of weak learner $h_j$ on $\boldsymbol{x}_i$, we have the optimization task

$$\begin{aligned} & \min_{\alpha_j, \xi_i} \sum_{j=1}^{T} \alpha_j + C \sum_{i=1}^{m} \xi_i \qquad (2.27) \\ s.t. \quad & y_i \sum\nolimits_{j=1}^{T} H_{i,j} \alpha_j + \xi_i \ge 1 \ (\forall i = 1, \ldots, m) \\ & \xi_i \ge 0 \ (\forall i = 1, \ldots, m) \\ & \alpha_j \ge 0 \ (\forall j = 1, \ldots, T) \ , \end{aligned}$$

or equivalently,

$$\begin{aligned} & \max_{\alpha_j, \xi_i, \rho} \rho - C' \sum_{i=1}^{m} \xi_i \qquad (2.28) \\ s.t. \quad & y_i \sum\nolimits_{j=1}^{T} H_{i,j} \alpha_j + \xi_i \ge \rho \ (\forall i = 1, \ldots, m) \\ & \sum\nolimits_{j=1}^{T} \alpha_j = 1 \\ & \xi_i \ge 0 \ (\forall i = 1, \ldots, m) \\ & \alpha_j \ge 0 \ (\forall j = 1, \ldots, T) \ , \end{aligned}$$

of which the dual form is

$$\begin{aligned} &\min_{w_i,\beta} \beta \\ s.t. \quad &\sum\nolimits_{i=1}^{m} w_i y_i H_{i,j} \leq \beta \; (\forall j = 1, \ldots, T) \\ &\sum\nolimits_{i=1}^{m} w_i = 1 \\ &w_i \in [0, C'] \; (\forall i = 1, \ldots, m) \; . \end{aligned} \tag{2.29}$$

A difficulty for the optimization task is that $T$ can be very large. Considering the final solution of the first linear programming, some $\alpha$ will be zero. One way to handle this problem is to find the smallest subset of all the columns; this can be done by *column generation* [Nash and Sofer, 1996]. Using the dual form, set $w_i = 1/m$ for the first column, and then, find the $j$th column that violates the constraint

$$\sum_{i=1}^{m} w_i y_i H_{i,j} \leq \beta \tag{2.30}$$

to the most. This is equivalent to maximizing $\sum_{i=1}^{m} w_i y_i H_{i,j}$; in other words, finding the weak learner $h_j$ with the smallest error under the weight distribution $\boldsymbol{w}$. When the solved $h_j$ does not violate any constraint, optimality is reached and the column generation process terminates. The whole procedure forms the LPBoost algorithm [Demiriz et al., 2002] summarized in Figure 2.10. The performance advantage of LPBoost against AdaBoost is not apparent [Demiriz et al., 2002], while it is observed that an improved version, entropy regularized LPBoost, often beats AdaBoost [Warmuth et al., 2008].

It is noteworthy that though the statistical view of boosting is well-accepted by the statistics community, it does not answer the question why AdaBoost seems resistant to overfitting. Moreover, the AdaBoost algorithm was designed as a classification algorithm for minimizing the misclassification error, while the statistical view focuses on the minimization of the surrogate loss function (or equivalently, probability estimation); these two problems are often very different. As indicated by Mease and Wyner [2008], in addition to the optimization aspect, a more comprehensive view should also consider the stagewise nature of the algorithm as well as the empirical variance reduction effect.

## 2.5 Multiclass Extention

In the previous sections we focused on AdaBoost for binary classification, i.e., $\mathcal{Y} = \{+1, -1\}$. In many classification tasks, however, an instance be-

```
**Input:** Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
           Base learning algorithm $\mathfrak{L}$;
           Parameter $\nu$.
           Number of learning rounds $T$.
**Process:**
$w_{1,i} = 1/m \ (\forall i = 1, \ldots, m)$.
$\beta_1 = 0$.
**for** $t = 1, \ldots$, T:
    $h_t = \mathfrak{L}(D, \boldsymbol{w})$;  % Train a learner $h_t$ from $D$ under $\boldsymbol{w}$
    **if** $\sum_{i=1}^{m} w_{t,i} y_i h_t(\boldsymbol{x}_i) \le \beta_t$ **then** $T = t - 1$; **break**;  % Check optimality
    $H_{i,t} = h_t(\boldsymbol{x}_i) \ (i = 1, \ldots, m)$ ;  % Fill a column
    $(\boldsymbol{w}_{t+1}, \beta_{t+1}) = \arg\min_{\boldsymbol{w},\beta} \beta$
        $s.t.$  $\sum_{i=1}^{m} w_i y_i h_j(\boldsymbol{x}_i) \le \beta \ (\forall j \le t)$
                $\sum_{i=1}^{m} w_i = 1$
                $w_i \in [0, \frac{1}{m\nu}] \ (\forall i = 1, \ldots, m)$
**end**
solve $\boldsymbol{\alpha}$ from the dual solution $(\boldsymbol{w}_{T+1}, \beta_{T+1})$;
**Output:** $H(\boldsymbol{x}) = \mathtt{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$
```

FIGURE 2.10: The LPBoost algorithm

longs to one of many instead of two classes. For example, a handwritten digit belongs to one of the ten classes, i.e., $\mathcal{Y} = \{0, \ldots, 9\}$. There are many alternative ways to extend AdaBoost for multiclass classification.

AdaBoost.M1 [Freund and Schapire, 1997] is a very straightforward extension, which is the same as the algorithm shown in Figure 2.2 except that the base learners now are multiclass learners instead of binary classifiers. This algorithm could not use binary classifiers, and has an overly strong constraint that every base learner has less than $1/2$ multiclass 0/1-loss.

SAMME [Zhu et al., 2006] is an improvement over AdaBoost.M1, which replaces the Line 6 of AdaBoost.M1 in Figure 2.2 by

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) + \ln(|\mathcal{Y}| - 1) \ . \tag{2.31}$$

This modification is derived from the minimization of multiclass exponential loss, and it was proved that, similar to the case of binary classification, optimizing the multiclass exponential loss approaches to the Bayes error rate, i.e.,

$$\mathtt{sign}(h^*(\boldsymbol{x})) = \arg\max_{y \in \mathcal{Y}} P(y|\boldsymbol{x}) \ , \tag{2.32}$$

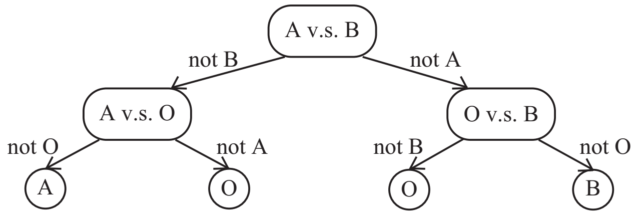


FIGURE 2.11: A directed acyclic graph that aggregates one-versus-one decomposition for classes of A, B and O.

where $h^*$ is the optimal solution to the multiclass exponential loss.

A commonly used solution to multiclass classification problem is to decompose the task into multiple binary classification problems. Popular decomposition schemes include **one-versus-rest** and **one-versus-one**. One-versus-rest decomposes a multiclass task of $|\mathcal{Y}|$ classes into $|\mathcal{Y}|$ binary classification tasks, where the $i$th task is to classify whether an instance belongs to the $i$th class or not. One-versus-one decomposes a multiclass task of $|\mathcal{Y}|$ classes into $\frac{|\mathcal{Y}|(|\mathcal{Y}|-1)}{2}$ binary classification tasks, where each task is to classify whether an instance belongs to, say, the $i$th class or the $j$th class.

AdaBoost.MH [Schapire and Singer, 1999] follows the one-versus-rest strategy. After training $|\mathcal{Y}|$ number of binary AdaBoost classifiers, the real-value output $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$ rather than the crisp classification of each AdaBoost classifier is used to identify the most probable class, that is,

$$H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} H_y(\boldsymbol{x}) \; , \tag{2.33}$$

where $H_y$ is the binary AdaBoost classifier that classifies the $y$th class from the rest.

AdaBoost.M2 [Freund and Schapire, 1997] follows the one-versus-one strategy, which minimizes a *pseudo-loss*. This algorithm is later generalized as AdaBoost.MR [Schapire and Singer, 1999] which minimizes a *ranking loss* motivated by the fact that the highest ranked class is more likely to be the correct class. Binary classifiers obtained by one-versus-one decomposition can also be aggregated by voting, pairwise coupling, directed acyclic graph, etc. [Hsu and Lin, 2002, Hastie and Tibshirani, 1998]. Voting and pairwise coupling are well-known, while Figure 2.11 illustrates the use of a directed acyclic graph.

## 2.6 Noise Tolerance

Real-world data are often noisy. The AdaBoost algorithm, however, was originally designed for clean data and has been observed to be very sensitive to noise. The noise sensitivity of AdaBoost is generally attributed to the exponential loss function (2.1) which specifies that if an instance were not classified as same as its given label, the weight of the instance will increase drastically. Consequently, when a training instance is associated with a wrong label, AdaBoost still tries to make the prediction resemble the given label, and thus degenerates the performance.

MadaBoost [Domingo and Watanabe, 2000] improves AdaBoost by depressing large instance weights. It is almost the same as AdaBoost except for the weight updating rule. Recall the weight updating rule of AdaBoost, i.e.,

$$\begin{aligned}\mathcal{D}_{t+1}(\boldsymbol{x}) &= \tfrac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} e^{-\alpha_t}, & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}) \\ e^{\alpha_t}, & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}) \end{cases} \\ &= \tfrac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times e^{-\alpha_t \cdot h_t(\boldsymbol{x}) \cdot f(\boldsymbol{x})} \\ &= \tfrac{\mathcal{D}_1(\boldsymbol{x})}{Z'_t} \times \textstyle\prod_{i=1}^{t} e^{-\alpha_i \cdot h_i(\boldsymbol{x}) \cdot f(\boldsymbol{x})} ,\end{aligned} \tag{2.34}$$

where $Z_t$ and $Z'_t$ are the normalization terms. It can be seen that, if the prediction on an instance is different from its given label for a number of rounds, the term $\prod_{i=1}^{t} e^{-\alpha_i \cdot h_i(\boldsymbol{x}) \cdot f(\boldsymbol{x})}$ will grow very large, pushing the instance to be classified according to the given label in the next round. To reduce the undesired dramatic increase of instance weights caused by noise, MadaBoost sets an upper limit on the weights:

$$\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_1(\boldsymbol{x})}{Z'_t} \times \min\left\{1, \prod_{i=1}^{t} e^{-\alpha_i \cdot h_i(\boldsymbol{x}) \cdot f(\boldsymbol{x})}\right\} , \tag{2.35}$$

where $Z'_t$ is the normalization term. By using this weight updating rule, the instance weights will not grow without bound.

FilterBoost [Bradley and Schapire, 2008] does not employ the exponential loss function used in AdaBoost, but adopts the log loss function (2.24). Similar to the derivation of AdaBoost in Section 2.2, we consider fitting an additive model to minimize the log loss function. At round $t$, denote the combined learner as $H_{t-1}$ and the classifier to be trained as $h_t$. Using the

Taylor expansion of the loss function, we have

$$\begin{aligned}\ell_{log}(H_{t-1}+h_t \mid \mathcal{D}) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[-\ln\frac{1}{1+e^{-f(\boldsymbol{x})(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x}))}}\right] \\ &\approx \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\ln(1+e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})})-\frac{f(\boldsymbol{x})h_t(\boldsymbol{x})}{1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}+\frac{e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{2(1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})})^2}\right] \\ &\approx \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[-\frac{f(\boldsymbol{x})h_t(\boldsymbol{x})}{1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}\right] ,\end{aligned} \tag{2.36}$$

by noticing that $f(\boldsymbol{x})^2 = 1$ and $h_t(\boldsymbol{x})^2 = 1$. To minimize the loss function, $h_t$ needs to satisfy

$$\begin{aligned}h_t &= \arg\min_h \ell_{log}(H_{t-1}+h \mid \mathcal{D}) \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{f(\boldsymbol{x})h(\boldsymbol{x})}{1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}\right] \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{f(\boldsymbol{x})h(\boldsymbol{x})}{Z_t(1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})})}\right] \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[f(\boldsymbol{x})h(\boldsymbol{x})\right] ,\end{aligned} \tag{2.37}$$

where $Z_t = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\frac{1}{1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}]$ is the normalization factor, and the weight updating rule is

$$\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})}{Z_t}\frac{1}{1+e^{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}} . \tag{2.38}$$

It is evident that with this updating rule, the increase of the instance weights is upper bounded by 1, similar to the weight depressing in MadaBoost, but smoother.

The BBM (Boosting-By-Majority) [Freund, 1995] algorithm was the first iterative boosting algorithm. Though it is noise tolerant [Aslam and Decatur, 1993], it requires the user to specify mysterious parameters in advance; excluding the requirement on unknown parameters motivated the development of AdaBoost. BrownBoost [Freund, 2001] is another adaptive version of BBM, which inherits BBM's noise tolerance property. Derived from the loss function of BBM, which is an accumulated binomial distribution, the loss function of BrownBoost is corresponding to the *Brownian motion process* [Gardiner, 2004], i.e.,

$$\ell_{bmp}(H_{t-1}+h_t \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[1-erf\left(\frac{f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+f(\boldsymbol{x})h_t(\boldsymbol{x})+c-t}{\sqrt{c}}\right)\right] , \tag{2.39}$$

where the parameter $c$ specifies the total *time* for the boosting procedure, $t$ is the current time which starts from zero and increases in each round, and

$erf(\cdot)$ is the error function

$$erf(a) = \frac{1}{\pi}\int_{-\infty}^{a} e^{-x^2} dx \ . \tag{2.40}$$

The loss function (2.39) can be expanded as

$$\begin{aligned}
\ell_{bmp}(H_{t-1} + h_t \mid \mathcal{D}) &\approx \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[ \begin{array}{l} 1 - erf(\frac{1}{\sqrt{c}}(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x}) + c - t) \\ -\dfrac{2}{c\pi} e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c} f(\boldsymbol{x})h_t(\boldsymbol{x}) \\ -\dfrac{4}{c^2\pi} e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c} f(\boldsymbol{x})^2 h_t(\boldsymbol{x})^2 \end{array} \right] \\
&\approx -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[ e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c} f(\boldsymbol{x})h_t(\boldsymbol{x}) \right] \ , \qquad (2.41)
\end{aligned}$$

and thus, the learner which minimizes the loss function is

$$\begin{aligned}
h_t &= \arg\min_h \ell_{bmp}(H_{t-1} + h \mid \mathcal{D}) \qquad (2.42) \\
&= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[ e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c} f(\boldsymbol{x})h(\boldsymbol{x}) \right] \\
&= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t} \left[ f(\boldsymbol{x})h(\boldsymbol{x}) \right] \ ,
\end{aligned}$$

where the weight updating rule is

$$\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})}{Z_t} e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c} \ , \tag{2.43}$$

and $Z_t$ is the normalization term. Noticing that the weighting function $e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})+c-t)^2/c}$ at here is quite different from that used in the boosting algorithms introduced above. When the classification margin $f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})$ equals the negative remaining time $-(c-t)$, the weight is set to the largest and will reduce as the margin goes either larger or smaller. This implies that BrownBoost/BBM "gives up" on some very hard training instances. With more learning rounds, $-(c-t)$ approaches 0. This implies that BrownBoost/BBM pushes the margin on most instances to be positive, while leaves alone the remaining hard instances that could be noise.

RobustBoost [Freund, 2009] is an improvement of BrownBoost, aiming at improving the noise tolerance ability by boosting the normalized classification margin, which is believed to be related to the generalization error (see Section 2.4.2). In other words, instead of minimizing the classification error, RobustBoost tries to minimize

$$\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[ \mathbb{I}\left( \frac{\sum_{t=1}^{T} \alpha_t f(\boldsymbol{x})h_t(\boldsymbol{x})}{\sum_{t=1}^{T} \alpha_t} \leq \theta \right) \right] \ , \tag{2.44}$$

where $\theta$ is the goal margin. For this purpose, the Brownian motion process in BrownBoost is changed to the mean-reverting *Ornstein-Uhlenbeck process* [Gardiner, 2004], with the loss function

$$\ell_{oup}(H_{t-1} + h_t \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[1 - erf\left(\frac{\tilde{m}(H_{t-1}(\boldsymbol{x}) + h_t(\boldsymbol{x})) - \mu(\frac{t}{c})}{\sigma(\frac{t}{c})}\right)\right] , \tag{2.45}$$

where $\tilde{m}(H)$ is the normalized margin of $H$, $0 \leq \frac{t}{c} \leq 1$, $\mu(\frac{t}{c}) = (\theta - 2\rho)e^{1-\frac{t}{c}} + 2\rho$ and $\sigma(\frac{t}{c})^2 = (\sigma_f^2 + 1)e^{2(1-\frac{t}{c})} - 1$ are respectively the mean and variance of the process, $\rho$, $\sigma_f$ as well as $\theta$ are parameters of the algorithm. By a similar derivation as that of BrownBoost, the weight updating rule of RobustBoost is

$$\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})}{Z_t} e^{-(f(\boldsymbol{x})H_{t-1}(\boldsymbol{x}) - \mu(\frac{t}{c}))^2/(2\sigma(\frac{t}{c})^2)} . \tag{2.46}$$

A major difference between the weighting functions (2.46) of RobustBoost from (2.43) of BrownBoost lies in the fact that $\mu(\frac{t}{c})$ approaches $\theta$ as $t$ approaches the total time $c$; thus, RobustBoost pushes the normalized classification margin to be larger than the goal margin $\theta$, while BrownBoost just pushes the classification margin to be larger than zero.

## 2.7 Further Readings

**Computational learning theory** studies some fundamental theoretical issues of learning. First introduced by Valiant [1984], the **PAC (Probably Approximately Correct)** framework models learning algorithms in a distribution free manner. Roughly speaking, for binary classification, a problem is *learnable* or *strongly learnable* if there exists an algorithm that outputs a learner $h$ in polynomial time such that for all $0 < \delta, \epsilon \leq 0.5$, $P(\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\mathbb{I}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})]] < \epsilon) \geq 1 - \delta$, and a problem is *weakly learnable* if there exists an algorithm that outputs a learner with error $0.5 - 1/p$ where $p$ is a polynomial in problem size and other parameters. Anthony and Biggs [1992], Kearns and Vazirani [1994] provide good introductions to computational learning theory.

In 1990, Schapire [1990] proved that strongly learnable problem class equals the weakly learnable problem class, an open problem raised by Kearns and Valiant [1989]. The proof is a construction, which is the first boosting algorithm. One year later, Freund developed the more efficient BBM algorithm, which was later published in [Freund, 1995]. Both algorithms, however, suffered from the practical deficiency that the error bounds of the base learners need to be known in advance. Later in 1995, Freund and Schapire [1995, 1997] developed the AdaBoost algorithm, which avoids the requirement on unknown parameters, thus named from *adaptive boosting*. The AdaBoost paper [Freund and Schapire, 1997] won its authors the *Gödel Prize* in 2003.

Understanding why AdaBoost seems resistant to overfitting is one of the most interesting open problems on boosting. Noticing that the concern is why AdaBoost *often* does not overfit, and it does not *never* overfit, e.g., Grove and Schuurmans [1998] showed that overfitting eventually occurs after enough learning rounds. Many interesting discussions can be found in the discussion part of [Friedman et al., 2000, Mease and Wyner, 2008].

Besides Breiman [1999], Harries [1999] also constructed an algorithm to show that the minimum margin is not crucial. Wang et al. [2008] introduced the *Emargin* and proved a new generalization error bound tighter than that based on the minimum margin. Gao and Zhou [2010b] showed that the minimum margin and Emargin are special cases of the $k$*th margin*; all of them are *single margins* that cannot measure the margin distribution. By considering exactly the same factors as Schapire et al. [1998], Gao and Zhou [2010b] proved a new generalization error bound based on the empirical Bernstein inequality [Maurer and Pontil, 2009]; this new generalization error bound is uniformly tighter than both the bounds of Schapire et al. [1998] and Breiman [1999], and thus defends the margin-based explanation against Breiman's doubt. Furthermore, Gao and Zhou [2010b] obtained an even tighter generalization error bound by considering the empirical *average margin* and *margin variance*.

In addition to the two most popular theoretical explanations, i.e., the margin explanation and the statistical view, there are also some other theoretical explanations to boosting. For example, Breiman [2004] proposed the population theory for boosting, Bickel et al. [2006] considered boosting as the Gauss-Southwell minimization of a loss function, etc. The **stability** of AdaBoost has also been studied [Kutin and Niyogi, 2002, Gao and Zhou, 2010a].

There are many empirical studies involving AdaBoost, e.g., [Bauer and Kohavi, 1999, Opitz and Maclin, 1999, Dietterich, 2000b]. The famous **bias-variance decomposition** [Geman et al., 1992] has been employed to empirically study why AdaBoost achieves excellent performance. This powerful tool breaks the expected error of a learning algorithm into the sum of three non-negative quantities, i.e., the **intrinsic noise**, the bias, and the variance. The **bias** measures how closely the average estimate of the learning algorithm is able to approximate the target, and the **variance** measures how much the estimate of the learning algorithm fluctuates for the different training sets of the same size. It has been observed that AdaBoost primarily reduces the bias though it is also able to reduce the variance [Bauer and Kohavi, 1999, Breiman, 1996a, Zhou et al., 2002b].

Ideal base learners for boosting are weak learners sufficiently strong to be boostable, since it is easy to underfit if the base learners are too weak, yet easy to overfit if the base learners are too strong. For binary classification, it is well-known that the exact requirement for weak learners is to be better than random guess. While for multi-class problems, it remains a mystery until the recent work by Mukherjee and Schapire [2010]. Notice that requiring base learners to be better than random guess is too weak for multi-class problems, yet requiring better than 50% accuracy is too stringent. Recently, **Conditional Random Fields (CRFs)** and **latent variable models** are also utilized as base learners for boosting [Dietterich et al., 2008, Hutchinson et al., 2011].

**Error Correcting Output Codes (ECOC)** [Dietterich and Bakiri, 1995] is an important way to extend binary learners to multi-class learners, which will be introduced in Section 4.6.1 of Chapter 4. Among the alternative ways of characterizing noise in data and how a learning algorithm is resistant to noise, the **statistical query model** [Kearns, 1998] is a PAC compliant theoretical model, in which a learning algorithm learns from queries of noisy expectation values of hypotheses. We call a boosting algorithm as a **SQ Boosting** if it efficiently boosts noise tolerant weak learners to strong learners. The noise tolerance of MadaBoost was proved by showing that it is a SQ Boosting, by assuming monotonic errors for the weak learners [Domingo and Watanabe, 2000]. Aslam and Decatur [1993] showed that BBM is also a SQ Boosting. In addition to algorithms introduced in Section 2.6, there are many other algorithms, such as GentleBoost [Friedman et al., 2000], trying to improve the robustness of AdaBoost. McDonald et al. [2003] reported an empirical comparison of AdaBoost, LogitBoost and BrownBoost on noisy data. A thorough comparison of robust AdaBoost variants is an important issue to be explored.