Chapter 7

AdaBoost

Zhi-Hua Zhou and Yang Yu

Contents

7.1 Introduction

Generalization ability, which characterizes how well the result learned from a given training data set can be applied to unseen new data, is the most central concept in machine learning. Researchers have devoted tremendous efforts to the pursuit of techniques that could lead to a learning system with strong generalization ability. One of the most successful paradigms is *ensemble learning* [32]. In contrast to ordinary machine learning approaches which try to generate *one* learner from training data, ensemble methods try to construct a *set* of *base learners* and combine them. Base learners are usually generated from training data by a *base learning algorithm* which can be a decision tree, a neural network, or other kinds of machine learning algorithms. Just like "many hands make light work," the generalization ability of an ensemble is usually significantly better than that of a single learner. Actually, ensemble methods are appealing mainly because they are able to boost *weak learners*, which are

slightly better than random guess, to *strong learners*, which can make very accurate predictions. So, "base learners" are also referred as "weak learners."

AdaBoost [9, 10] is one of the most influential ensemble methods. It took birth from the answer to an interesting question posed by Kearns and Valiant in 1988. That is, whether two complexity classes, *weakly learnable* and *strongly learnable* problems, are equal. If the answer to the question is positive, a weak learner that performs just slightly better than random guess can be "boosted" into an arbitrarily accurate strong learner. Obviously, such a question is of great importance to machine learning. Schapire [21] found that the answer to the question is "yes," and gave a proof by construction, which is the first *boosting* algorithm. An important practical deficiency of this algorithm is the requirement that the error bound of the base learners be known ahead of time, which is usually unknown in practice. Freund and Schapire [9] then proposed an adaptive boosting algorithm, named AdaBoost, which does not require those unavailable information. It is evident that AdaBoost was born with theoretical significance, which has given rise to abundant research on theoretical aspects of ensemble methods in communities of machine learning and statistics. It is worth mentioning that for their AdaBoost paper [9], Schapire and Freund won the Godel Prize, which is one of the most prestigious awards in theoretical computer science, in the year 2003.

AdaBoost and its variants have been applied to diverse domains with great success, owing to their solid theoretical foundation, accurate prediction, and great simplicity (Schapire said it needs only "just 10 lines of code"). For example, Viola and Jones [27] combined AdaBoost with a cascade process for face detection. They regarded rectangular features as weak learners, and by using AdaBoost to weight the weak learners, they got very intuitive features for face detection. In order to get high accuracy as well as high efficiency, they used a cascade process (which is beyond the scope of this chapter). As a result, they reported a very strong face detector: On a 466 MHz machine, face detection on a 384×288 image costs only 0.067 second, which is 15 times faster than state-of-the-art face detectors at that time but with comparable accuracy. This face detector has been recognized as one of the most exciting breakthroughs in computer vision (in particular, face detection) during the past decade. It is not strange that "boosting" has become a buzzword in computer vision and many other application areas.

In the rest of this chapter, we will introduce the algorithm and implementations, and give some illustrations on how the algorithm works. For readers who are eager to know more, we will introduce some theoretical results and extensions as advanced topics.

7.2 The Algorithm

7.2.1 Notations

We first introduce some notations that will be used in the rest of the chapter. Let $\mathcal{X}$ denote the instance space, or in other words, feature space. Let $\mathcal{Y}$ denote the set of labels that express the underlying concepts which are to be learned. For example, we

let $\mathcal{Y} = \{-1, +1\}$ for binary classification. A training set D consists of m instances whose associated labels are observed, i.e., $D = \{(\boldsymbol{x}_i, y_i)\}$ $(i \in \{1, \ldots, m\})$, while the label of a test instance is unknown and thus to be predicted. We assume both training and test instances are drawn independently and identically from an underlying distribution $\mathcal{D}$.

After training on a training data set D, a learning algorithm $\mathcal{L}$ will output a hypothesis h, which is a mapping from $\mathcal{X}$ to $\mathcal{Y}$, or called as a *classifier*. The learning process can be regarded as picking the best hypothesis from a hypothesis space, where the word "best" refers to a loss function. For classification, the loss function can naturally be 0/1-loss,

$$loss_{0/1}(h \mid \boldsymbol{x}) = \boldsymbol{I}[h(\boldsymbol{x}) \neq y]$$

where $\boldsymbol{I}[\cdot]$ is the indication function which outputs 1 if the inner expression is true and 0 otherwise, which means that one error is counted if an instance is wrongly classified. In this chapter 0/1-loss is used by default, but it is noteworthy that other kinds of loss functions can also be used in boosting.

7.2.2 A General Boosting Procedure

Boosting is actually a family of algorithms, among which the AdaBoost algorithm is the most influential one. So, it may be easier by starting from a general boosting procedure.

Suppose we are dealing with a binary classification problem, that is, we are trying to classify instances as *positive* and *negative*. Usually we assume that there exists an unknown target concept, which correctly assigns "positive" labels to instances belonging to the concept and "negative" labels to others. This unknown target concept is actually what we want to learn. We call this target concept *ground-truth*. For a binary classification problem, a classifier working by random guess will have 50% 0/1-loss.

Suppose we are unlucky and only have a *weak classifier* at hand, which is only slightly better than random guess on the underlying instance distribution $\mathcal{D}$, say, it has 49% 0/1-loss. Let's denote this weak classifier as h_1. It is obvious that h_1 is not what we want, and we will try to improve it. A natural idea 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, for example, it focuses more on the instances wrongly classified by h_1 (we will explain how to generate $\mathcal{D}'$ in the next section). We can train a classifier h_2 from $\mathcal{D}'$. Again, suppose we are unlucky and h_2 is also a weak classifier. Since $\mathcal{D}'$ was derived from $\mathcal{D}$, if $\mathcal{D}'$ satisfies some condition, h_2 will be able to achieve a better performance than h_1 on some places in $\mathcal{D}$ where h_1 does not work well, without scarifying the places where h_1 performs well. Thus, 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 be able to achieve less loss than that achieved by h_1. By repeating the above process, we can expect to get a combined classifier which has very small (ideally, zero) 0/1-loss on $\mathcal{D}$.

Input: Instance distribution $\mathcal{D}$;
Base learning algorithm L;
Number of learning rounds T.
Process:
1. $\mathcal{D}_1 = \mathcal{D}$. % Initialize distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = L(\mathcal{D}_t)$; % Train a weak learner from distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of h_t
5. $\mathcal{D}_{t+1} = AdjustDistribution\ (\mathcal{D}_t, \epsilon_t)$
6. **end**
Output: $H(\boldsymbol{x}) = CombineOutputs(\{h_t(\boldsymbol{x})\})$

Figure 7.1 A general boosting procedure.

Briefly, boosting works by training a set of classifiers sequentially and combining them for prediction, where the later classifiers focus more on the mistakes of the earlier classifiers. Figure 7.1 summarizes the general boosting procedure.

7.2.3 The AdaBoost Algorithm

Figure 7.1 is not a real algorithm since there are some undecided parts such as $AdjustDistribution$ and $CombineOutputs$. The AdaBoost algorithm can be viewed as an instantiation of the general boosting procedure, which is summarized in Figure 7.2.

Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm L;
Number of learning rounds T.
Process:
1. $\mathcal{D}_1(i) = 1/m$. % Initialize the weight distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = L(D, \mathcal{D}_t)$; % Train a learner h_t from D using distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of h_t
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % Determine the weight of h_t
7. $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$
$= \frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$ % Update the distribution, where
% Z_t is a normalization factor which
% enables $\mathcal{D}_{t+1}$ to be distribution
8. **end**

Output: $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$

Figure 7.2 The AdaBoost algorithm.

Now we explain the details.[1] AdaBoost generates a sequence of hypotheses and combines them with weights, which can be regarded as an *additive* weighted combination in the form of

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

From this view, AdaBoost actually solves two problems, that is, how to generate the hypotheses h_t's and how to determine the proper weights α_t's.

In order to have a highly efficient error reduction process, we try to minimize an exponential loss

$$loss_{\exp}(h) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},y}[e^{-yh(\boldsymbol{x})}]$$

where $yh(\boldsymbol{x})$ is called as the *classification margin* of the hypothesis.

Let's consider one round in the boosting process. Suppose a set of hypotheses as well as their weights have already been obtained, and let H denote the combined hypothesis. Now, one more hypothesis h will be generated and is to be combined with H to form $H + \alpha h$. The loss after the combination will be

$$loss_{\exp}(H + \alpha h) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},y}[e^{-y(H(\boldsymbol{x})+\alpha h(\boldsymbol{x}))}]$$

The loss can be decomposed to each instance, which is called pointwise loss, as

$$loss_{\exp}(H + \alpha h \mid \boldsymbol{x}) = \mathbb{E}_{y}[e^{-y(H(\boldsymbol{x})+\alpha h(\boldsymbol{x}))} \mid \boldsymbol{x}]$$

Since y and $h(\boldsymbol{x})$ must be $+1$ or -1, we can expand the expectation as

$$loss_{\exp}(H + \alpha h \mid \boldsymbol{x}) = e^{-yH(\boldsymbol{x})}\left(e^{-\alpha}P(y = h(\boldsymbol{x}) \mid \boldsymbol{x}) + e^{\alpha}P(y \neq h(\boldsymbol{x}) \mid \boldsymbol{x})\right)$$

Suppose we have already generated h, and thus the weight α that minimizes the loss can be found when the derivative of the loss equals zero, that is,

$$\begin{aligned}\frac{\partial loss_{\exp}(H + \alpha h \mid \boldsymbol{x})}{\partial \alpha} &= e^{-yH(\boldsymbol{x})}\left(-e^{-\alpha}P(y = h(\boldsymbol{x}) \mid \boldsymbol{x}) + e^{\alpha}P(y \neq h(\boldsymbol{x}) \mid \boldsymbol{x})\right)\\ &= 0\end{aligned}$$

and the solution is

$$\alpha = \frac{1}{2}\ln\frac{P(y = h(\boldsymbol{x}) \mid \boldsymbol{x})}{P(y \neq h(\boldsymbol{x}) \mid \boldsymbol{x})} = \frac{1}{2}\ln\frac{1 - P(y \neq h(\boldsymbol{x}) \mid \boldsymbol{x})}{P(y \neq h(\boldsymbol{x}) \mid \boldsymbol{x})}$$

By taking an expectation over $\boldsymbol{x}$, that is, solving $\frac{\partial loss_{\exp}(H+\alpha h)}{\partial \alpha} = 0$, and denoting $\epsilon = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[y \neq h(\boldsymbol{x})]$, we get

$$\alpha = \frac{1}{2}\ln\frac{1 - \epsilon}{\epsilon}$$

which is the way of determining α_t in AdaBoost.

[1] Here we explain the AdaBoost algorithm from the view of [11] since it is easier to understand than the original explanation in [9].

Now let's consider how to generate h. Given a base learning algorithm, AdaBoost invokes it to produce a hypothesis from a particular instance distribution. So, we only need to consider what hypothesis is desired for the next round, and then generate an instance distribution to achieve this hypothesis.

We can expand the pointwise loss to second order about $h(\boldsymbol{x}) = 0$, when fixing $\alpha = 1$,

$$\begin{aligned} loss_{\exp}(H + h \mid \boldsymbol{x}) &\approx \mathbb{E}_y[e^{-yH(\boldsymbol{x})}(1 - yh(\boldsymbol{x}) + y^2h(\boldsymbol{x})^2/2) \mid \boldsymbol{x}] \\ &= \mathbb{E}_y[e^{-yH(\boldsymbol{x})}(1 - yh(\boldsymbol{x}) + 1/2) \mid \boldsymbol{x}] \end{aligned}$$

since $y^2 = 1$ and $h(\boldsymbol{x})^2 = 1$.

Then a perfect hypothesis is

$$\begin{aligned} h^*(\boldsymbol{x}) &= \arg\min_h loss_{\exp}(H + h \mid \boldsymbol{x}) = \arg\max_h \mathbb{E}_y[e^{-yH(\boldsymbol{x})}yh(\boldsymbol{x}) \mid \boldsymbol{x}] \\ &= \arg\max_h e^{-H(\boldsymbol{x})}P(y = 1 \mid \boldsymbol{x}) \cdot 1 \cdot h(\boldsymbol{x}) + e^{H(\boldsymbol{x})}P(y = -1 \mid \boldsymbol{x}) \cdot (-1) \cdot h(\boldsymbol{x}) \end{aligned}$$

Note that $e^{-yH(\boldsymbol{x})}$ is a constant in terms of $h(\boldsymbol{x})$. By normalizing the expectation as

$$h^*(\boldsymbol{x}) = \arg\max_h \frac{e^{-H(\boldsymbol{x})}P(y = 1 \mid \boldsymbol{x}) \cdot 1 \cdot h(\boldsymbol{x}) + e^{H(\boldsymbol{x})}P(y = -1 \mid \boldsymbol{x}) \cdot (-1) \cdot h(\boldsymbol{x})}{e^{-H(\boldsymbol{x})}P(y = 1 \mid \boldsymbol{x}) + e^{H(\boldsymbol{x})}P(y = -1 \mid \boldsymbol{x})}$$

we can rewrite the expectation using a new term $w(\boldsymbol{x}, y)$, which is drawn from $e^{-yH(\boldsymbol{x})}P(y \mid \boldsymbol{x})$, as

$$h^*(\boldsymbol{x}) = \arg\max_h \mathbb{E}_{w(\boldsymbol{x},y)\sim e^{-yH(\boldsymbol{x})}P(y|\boldsymbol{x})}[yh(\boldsymbol{x}) \mid \boldsymbol{x}]$$

Since $h^*(\boldsymbol{x})$ must be $+1$ or -1, the solution to the optimization is that $h^*(\boldsymbol{x})$ holds the same sign with $y|\boldsymbol{x}$, that is,

$$\begin{aligned} h^*(\boldsymbol{x}) &= \mathbb{E}_{w(\boldsymbol{x},y)\sim e^{-yH(\boldsymbol{x})}P(y|\boldsymbol{x})}[y \mid \boldsymbol{x}] \\ &= P_{w(\boldsymbol{x},y)\sim e^{-yH(\boldsymbol{x})}P(y|\boldsymbol{x})}(y = 1 \mid \boldsymbol{x}) - P_{w(\boldsymbol{x},y)\sim e^{-yH(\boldsymbol{x})}P(y|\boldsymbol{x})}(y = -1 \mid \boldsymbol{x}) \end{aligned}$$

As can be seen, h^* simply performs the optimal classification of $\boldsymbol{x}$ under the distribution $e^{-yH(\boldsymbol{x})}P(y \mid \boldsymbol{x})$. Therefore, $e^{-yH(\boldsymbol{x})}P(y \mid \boldsymbol{x})$ is the desired distribution for a hypothesis minimizing 0/1-loss.

So, when the hypothesis $h(\boldsymbol{x})$ has been learned and $\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}$ has been determined in the current round, the distribution for the next round should be

$$\begin{aligned} \mathcal{D}_{t+1}(\boldsymbol{x}) &= e^{-y(H(\boldsymbol{x})+\alpha h(\boldsymbol{x}))}P(y \mid \boldsymbol{x}) = e^{-yH(\boldsymbol{x})}P(y \mid \boldsymbol{x}) \cdot e^{-\alpha yh(\boldsymbol{x})} \\ &= \mathcal{D}_t(\boldsymbol{x}) \cdot e^{-\alpha yh(\boldsymbol{x})} \end{aligned}$$

which is the way of updating instance distribution in AdaBoost.

But, why optimizing the exponential loss works for minimizing the 0/1-loss? Actually, we can see that

$$h^*(\boldsymbol{x}) = \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},y}[e^{-yh(\boldsymbol{x})} \mid \boldsymbol{x}] = \frac{1}{2}\ln\frac{P(y = 1 \mid \boldsymbol{x})}{P(y = -1 \mid \boldsymbol{x})}$$

and therefore we have

$$sign(h^*(\boldsymbol{x})) = \arg\max_y P(y|\boldsymbol{x})$$

which implies that the optimal solution to the exponential loss achieves the minimum Bayesian error for the classification problem. Moreover, we can see that the function h^* which minimizes the exponential loss is the logistic regression model up to a factor 2. So, by ignoring the factor $1/2$, AdaBoost can also be viewed as fitting an additive logistic regression model.

It is noteworthy that the data distribution is not known in practice, and the AdaBoost algorithm works on a given training set with finite training examples. Therefore, all the expectations in the above derivations are taken on the training examples, and the weights are also imposed on training examples. For base learning algorithms that cannot handle weighted training examples, a resampling mechanism, which samples training examples according to desired weights, can be used instead.

7.3 Illustrative Examples

In this section, we demonstrate how the AdaBoost algorithm works, from an illustration on a toy problem to real data sets.

7.3.1 Solving XOR Problem

We consider an artificial data set in a two-dimensional space, plotted in Figure 7.3(a). There are only four instances, that is,

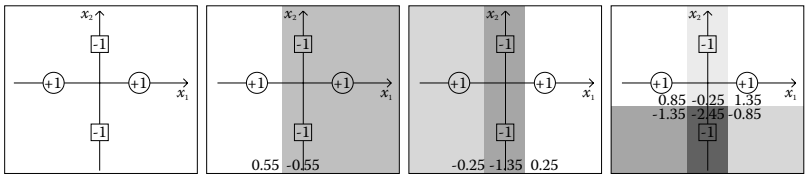
$$\left\{\begin{array}{l}(\boldsymbol{x}_1 = (0, +1), y_1 = +1)\\ (\boldsymbol{x}_2 = (0, -1), y_2 = +1)\\ (\boldsymbol{x}_3 = (+1, 0), y_3 = -1)\\ (\boldsymbol{x}_4 = (-1, 0), y_4 = -1)\end{array}\right\}$$

This is the XOR problem. The two classes cannot be separated by a linear classifier which corresponds to a line on the figure.

Suppose we have a base learning algorithm which tries to select the best of the following eight functions. Note that none of them is perfect. For equally good functions, the base learning algorithm will pick one function from them randomly.

$$h_1(\boldsymbol{x}) = \left\{\begin{array}{ll}+1, & \text{if } (x_1 > -0.5)\\ -1, & \text{otherwise}\end{array}\right. \quad h_2(\boldsymbol{x}) = \left\{\begin{array}{ll}-1, & \text{if } (x_1 > -0.5)\\ +1, & \text{otherwise}\end{array}\right.$$

$$h_3(\boldsymbol{x}) = \left\{\begin{array}{ll}+1, & \text{if } (x_1 > +0.5)\\ -1, & \text{otherwise}\end{array}\right. \quad h_4(\boldsymbol{x}) = \left\{\begin{array}{ll}-1, & \text{if } (x_1 > +0.5)\\ +1, & \text{otherwise}\end{array}\right.$$



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

Figure 7.3 AdaBoost on the XOR problem.

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

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

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

Now we track how AdaBoost works:

1. The first step is to invoke the base learning algorithm on the original data. h_2, h_3, h_5, and h_8 all have 0.25 classification errors. Suppose h_2 is picked as the first base learner. One instance, $\boldsymbol{x}_1$, is wrongly classified, so the error is $1/4 = 0.25$. The weight of h_2 is $0.5 \ln 3 \approx 0.55$. Figure 7.3(b) visualizes the classification, where the shadowed area is classified as negative (-1) and the weights of the classification, 0.55 and -0.55, are displayed.
2. The weight of $\boldsymbol{x}_1$ is increased and the base learning algorithm is invoked again. This time h_3, h_5, and h_8 have equal errors. Suppose h_3 is picked, of which the weight is 0.80. Figure 7.3(c) shows the combined classification of h_2 and h_3 with their weights, where different gray levels are used for distinguishing negative areas according to classification weights.
3. The weight of $\boldsymbol{x}_3$ is increased, and this time only h_5 and h_8 equally have the lowest errors. Suppose h_5 is picked, of which the weight is 1.10. Figure 7.3(d) shows the combined classification of h_2, h_3, and h_8. If we look at the sign of classification weights in each area in Figure 7.3(d), all the instances are correctly classified. Thus, by combining the imperfect linear classifiers, AdaBoost has produced a nonlinear classifier which has zero error.

7.3.2 Performance on Real Data

We evaluate the AdaBoost algorithm on 56 data sets from the UCI Machine Learning Repository,[2] which covers a broad range of real-world tasks. We use the Weka (will be introduced in Section 7.6) implementation of AdaBoost.M1 using reweighting with

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

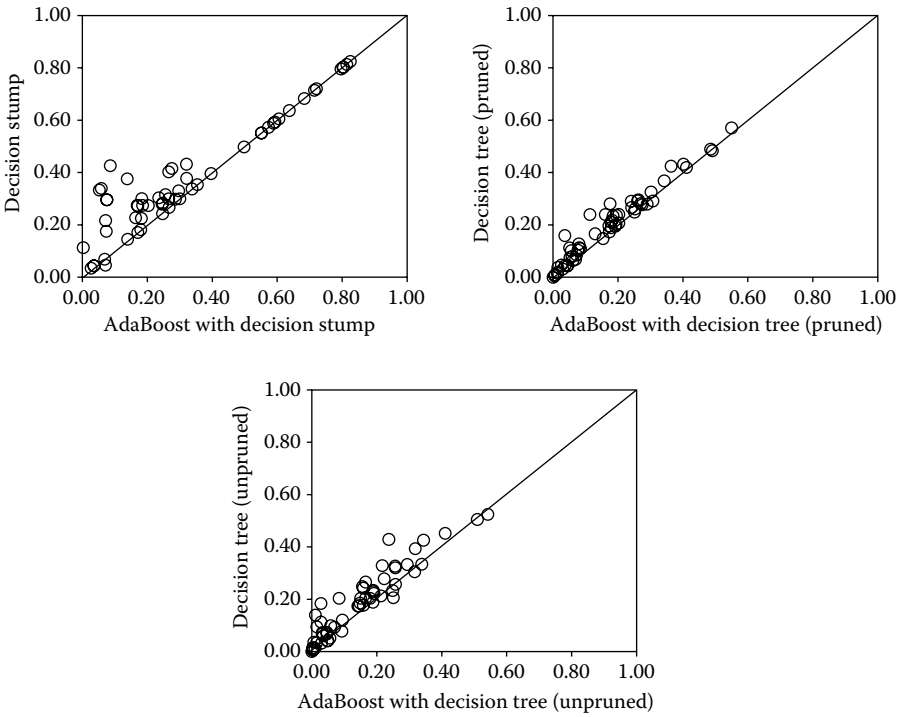


Figure 7.4 Comparison of predictive errors of AdaBoost against decision stump, pruned, and unpruned single decision trees on 56 UCI data sets.

50 base learners. Almost all kinds of learning algorithms can be taken as base learning algorithms, such as decision trees, neural networks, and so on. Here, we have tried three base learning algorithms, including decision stump, pruned, and unpruned J4.8 decision trees (Weka implementation of C4.5).

We plot the comparison results in Figure 7.4, where each circle represents a data set and locates according to the predictive errors of the two compared algorithms. In each plot of Figure 7.4, the diagonal line indicates where the two compared algorithms have identical errors. It can be observed that AdaBoost often outperforms its base learning algorithm, with a few exceptions on which it degenerates the performance.

The famous *bias-variance decomposition* [12] has been employed to empirically study why AdaBoost achieves excellent performance [2, 3, 34]. This powerful tool breaks the expected error of a learning approach into the sum of three nonnegative quantities, that is, the intrinsic noise, the bias, and the variance. The bias measures how closely the average estimate of the learning approach is able to approximate the target, and the variance measures how much the estimate of the learning approach fluctuates for the different training sets of the same size. It has been observed [2,3,34] that AdaBoost primarily reduces the bias but it is also able to reduce the variance.

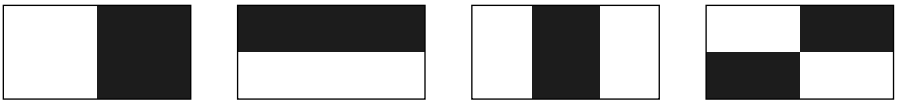


Figure 7.5 Four feature masks to be applied to each rectangle.

7.4 Real Application

Viola and Jones [27] combined AdaBoost with a cascade process for face detection. As the result, they reported that on a 466 MHz machine, face detection on a 384×288 image costs only 0.067 seconds, which is almost 15 times faster than state-of-the-art face detectors at that time but with comparable accuracy. This face detector has been recognized as one of the most exciting breakthroughs in computer vision (in particular, face detection) during the past decade. In this section, we briefly introduce how AdaBoost works in the Viola-Jones face detector.

Here the task is to locate all possible human faces in a given image. An image is first divided into subimages, say 24×24 squares. Each subimage is then represented by a feature vector. To make the computational process efficient, very simple features are used. All possible rectangles in a subimage are examined. On every rectangle, four features are extracted using the masks shown in Figure 7.5. With each mask, the sum of pixels' gray level in white areas is subtracted by the sum of those in dark areas, which is regarded as a feature. Thus, by a 24×24 splitting, there are more than 1 million features, but each of the features can be calculated very fast.

Each feature is regarded as a weak learner, that is,

$$h_{i,p,\theta}(\boldsymbol{x}) = \boldsymbol{I}[px_i \leq p\theta] \quad (p \in \{+1, -1\})$$

where x_i is the value of $\boldsymbol{x}$ at the i-th feature.

The base learning algorithm tries to find the best weak classifier h_{i^*,p^*,θ^*} that minimizes the classification error, that is,

$$(i^*, p^*, \theta^*) = \arg\min_{i,p,\theta} \mathbb{E}_{(\boldsymbol{x},y)} \boldsymbol{I}[h_{i,p,\theta}(\boldsymbol{x}) \neq y]$$

Face rectangles are regarded as positive examples, as shown in Figure 7.6, while rectangles that do not contain any face are regarded as negative examples. Then, the AdaBoost process is applied and it will return a few weak learners, each corresponds to one of the over 1 million features. Actually, the AdaBoost process can be regarded as a feature selection tool here.

Figure 7.7 shows the first two selected features and their position relative to a human face. It is evident that these two features are intuitive, where the first feature measures how the intensity of the eye areas differ from that of the lower areas, while

Figure 7.6 Positive training examples [27].

the second feature measures how the intensity of the two eye areas differ from the area between two eyes.

Using the selected features in order, an extremely imbalanced decision tree is built, which is called *cascade* of classifiers, as illustrated in Figure 7.8.

The parameter θ is adjusted in the cascade such that, at each tree node, branching into "not a face" means that the image is really not a face. In other words, the false negative rate is minimized. This design owes to the fact that a nonface image is easier to be recognized, and it is possible to use a few features to filter out a lot of candidate image rectangles, which endows the high efficiency. It was reported [27] that 10 features per subimage are examined in average. Some test results of the Viola-Jones face detector are shown in Figure 7.9.

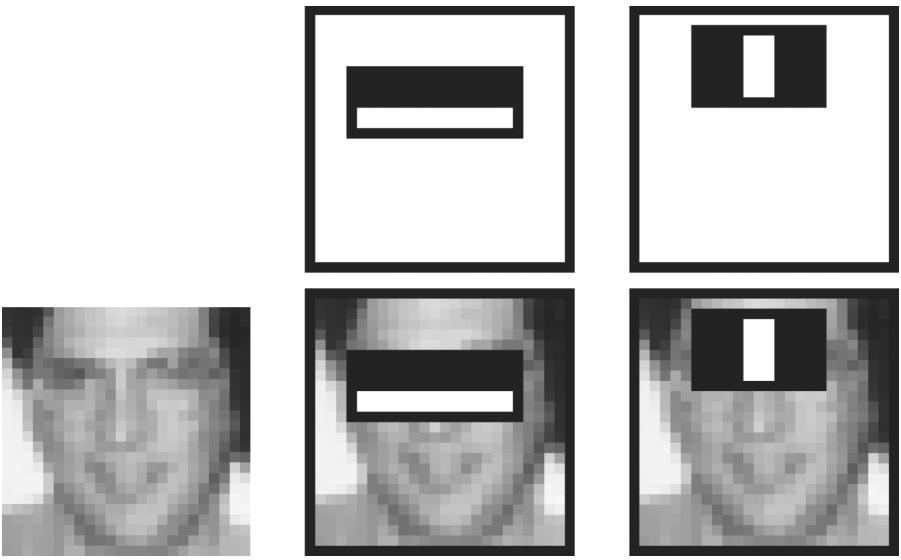


Figure 7.7 Selected features [27].

7.5 Advanced Topics

7.5.1 Theoretical Issues

Computational learning theory studies some fundamental theoretical issues of machine learning. First introduced by Valiant in 1984 [25], the *Probably Approximately Correct* (PAC) 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 hypothesis h in polynomial time

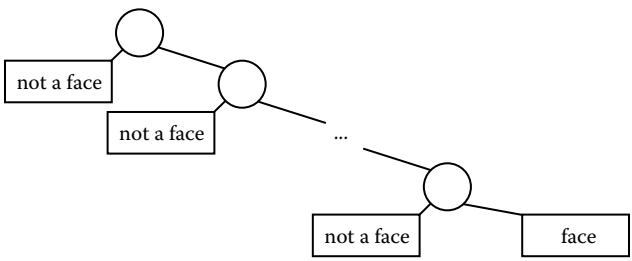


Figure 7.8 A cascade of classifiers.

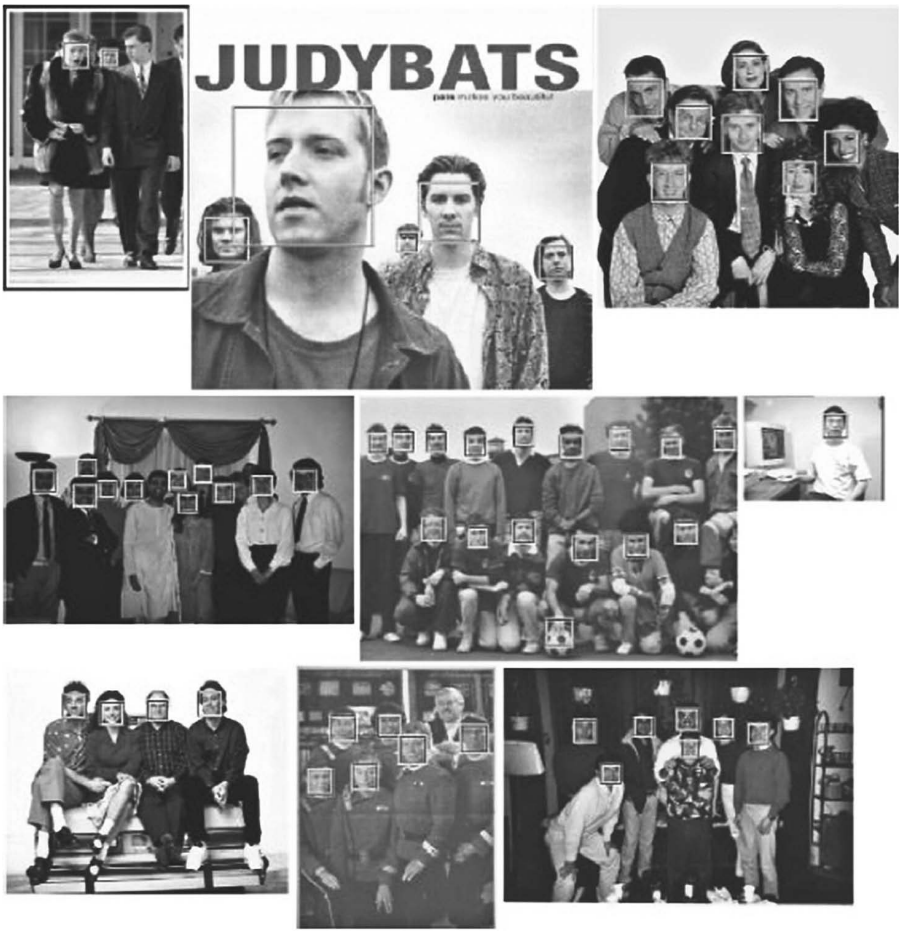


Figure 7.9 Outputs of the Viola-Jones face detector on a number of test images [27].

such that for all $0 < \delta, \epsilon \leq 0.5$,

$$P\left(\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},y}\left[\boldsymbol{I}\left[h(\boldsymbol{x}) \neq y\right]\right] < \epsilon\right) \geq 1 - \delta$$

and a problem is *weakly learnable* if the above holds for all $0 < \delta \leq 0.5$ but only when ϵ is slightly smaller than 0.5 (or in other words, h is only slightly better than random guess).

In 1988, Kearns and Valiant [15] posed an interesting question, that is, whether the strongly learnable problem class equals the weakly learnable problem class. This question is of fundamental importance, since if the answer is "yes," any weak learner is potentially able to be boosted to a strong learner. In 1989, Schapire [21] proved that the answer is really "yes," and the proof he gave is a construction, which is the first

boosting algorithm. One year later, Freund [7] developed a more efficient algorithm. Both algorithms, however, suffered from the practical deficiency that the error bound of the base learners need to be known ahead of time, which is usually unknown in practice. Later, in 1995, Freund and Schapire [9] developed the AdaBoost algorithm, which is effective and efficient in practice.

Freund and Schapire [9] proved that, if the base learners of AdaBoost have errors $\epsilon_1, \epsilon_2, \cdots, \epsilon_T$, the error of the final combined learner, ϵ, is upper bounded as

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

where $\gamma_t = 0.5 - \epsilon_t$. It can be seen that AdaBoost reduces the error exponentially fast. Also, it can be derived that, to achieve an error less than ϵ, the round T is upper bounded as

$$T \leq \left\lceil \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} \right\rceil$$

where it is assumed that $\gamma = \gamma_1 = \gamma_2 = \cdots = \gamma_T$.

In practice, however, all the operations of AdaBoost can only be carried out on training data D, that is,

$$\epsilon_D = \mathbb{E}_{\boldsymbol{x}\sim D,y}\boldsymbol{I}[H(\boldsymbol{x}) \neq y]$$

and thus the errors are training errors, while the generalization error, that is, the error over instance distribution $\mathcal{D}$

$$\epsilon_{\mathcal{D}} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},y}\boldsymbol{I}[H(\boldsymbol{x}) \neq y]$$

is of more interest.

The initial analysis [9] showed that the generalization error of AdaBoost is upper bounded as

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

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

The above bound suggests that in order to achieve a good generalization ability, it is necessary to constrain the complexity of base learners as well as the number of learning rounds; otherwise AdaBoost will overfit. However, empirical studies show that AdaBoost often does not overfit, that is, its test error often tends to decrease even after the training error reaches zero, even after a large number of rounds, such as 1000.

For example, Schapire et al. [22] plotted the performance of AdaBoost on the *letter* data set from UCI Machine Learning Repository, as shown in Figure 7.10 (left), where the higher curve is test error while the lower one is training error. It can be observed that AdaBoost achieves zero training error in less than 10 rounds but the generalization error keeps on reducing. This phenomenon seems to counter Occam's

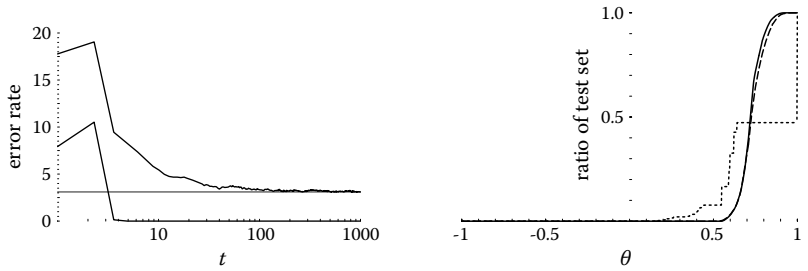


Figure 7.10 Training and test error (left) and margin distribution (right) of AdaBoost on the *letter* data set [22].

Razor, that is, nothing more than necessary should be done, which is one of the basic principles in machine learning.

Many researchers have studied this phenomena, and several theoretical explanations have been given, for example, [11]. Schapire et al. [22] introduced the *margin*-based explanation. They argued that AdaBoost is able to increase the margin even after the training error reaches zero, and thus it does not overfit even after a large number of rounds. The classification margin of h on $\boldsymbol{x}$ is defined as $yh(\boldsymbol{x})$, and that of $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$ is defined as

$$yH(\boldsymbol{x}) = \frac{\sum_{t=1}^{T} \alpha_t y h_t(\boldsymbol{x})}{\sum_{t=1}^{T} \alpha_t}$$

Figure 7.10 (right) plots the distribution of $yH(\boldsymbol{x}) \leq \theta$ for different values of θ. It was proved in [22] that the generalization error is upper bounded as

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

with probability at least $1 - \delta$. This bound qualitatively explains that when other variables in the bound are fixed, the larger the margin, the smaller the generalization error.

However, this margin-based explanation was challenged by Brieman [4]. Using *minimum margin* ϱ,

$$\varrho = \min_{\boldsymbol{x}\in D} yH(\boldsymbol{x})$$

Breiman proved a generalization error bound is tighter than the above one using minimum margin. Motivated by the tighter bound, the arc-gv algorithm, which is a variant of AdaBoost, was proposed to maximize the minimum margin directly, by

updating α_t according to

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

Interestingly, the minimum margin of arc-gv is uniformly better than that of AdaBoost, but the test error of arc-gv increases drastically on all tested data sets [4]. Thus, the margin theory for AdaBoost was almost sentenced to death.

In 2006, Reyzin and Schapire [20] reported an interesting finding. It is well-known that the bound of the generalization error is associated with margin, the number of rounds, and the complexity of base learners. When comparing arc-gv with AdaBoost, Breiman [4] tried to control the complexity of base learners by using decision trees with the same number of leaves, but Reyzin and Schapire found that these are trees with 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 7.11 (top) shows the difference of depth of the trees generated by the two algorithms on the *breast cancer* data set from UCI Machine Learning Repository. Although 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 repeated Breiman's experiments by using decision stump, which has only one leaf and therefore is with a fixed complexity, and found that the margin distribution of AdaBoost is actually better than that of arc-gv, as illustrated in Figure 7.11 (bottom).

Recently, Wang et al. [28] introduced *equilibrium margin* and proved a new bound tighter than that obtained by using minimum margin, which suggests that the minimum margin may not be crucial for the generalization error of AdaBoost. It will be interesting to develop an algorithm that maximizes equilibrium margin directly, and to see whether the test error of such an algorithm is smaller than that of AdaBoost, which remains an open problem.

7.5.2 Multiclass AdaBoost

In the previous sections we focused on AdaBoost for binary classification, that is, $\mathcal{Y} = \{+1, -1\}$. In many classification tasks, however, an instance belongs to one of many instead of two classes. For example, a handwritten number belongs to 1 of 10 classes, that is, $\mathcal{Y} = \{0, \ldots, 9\}$. There is more than one way to deal with a multiclass classification problem.

AdaBoost.M1 [9] is a very direct extension, which is as same as the algorithm shown in Figure 7.2, except that now the base learners are multiclass learners instead of binary classifiers. This algorithm could not use binary base classifiers, and requires every base learner have less than $1/2$ multiclass 0/1-loss, which is an overstrong constraint.

SAMME [35] is an improvement over AdaBoost.M1, which replaces Line 5 of AdaBoost.M1 in Figure 7.2 by

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

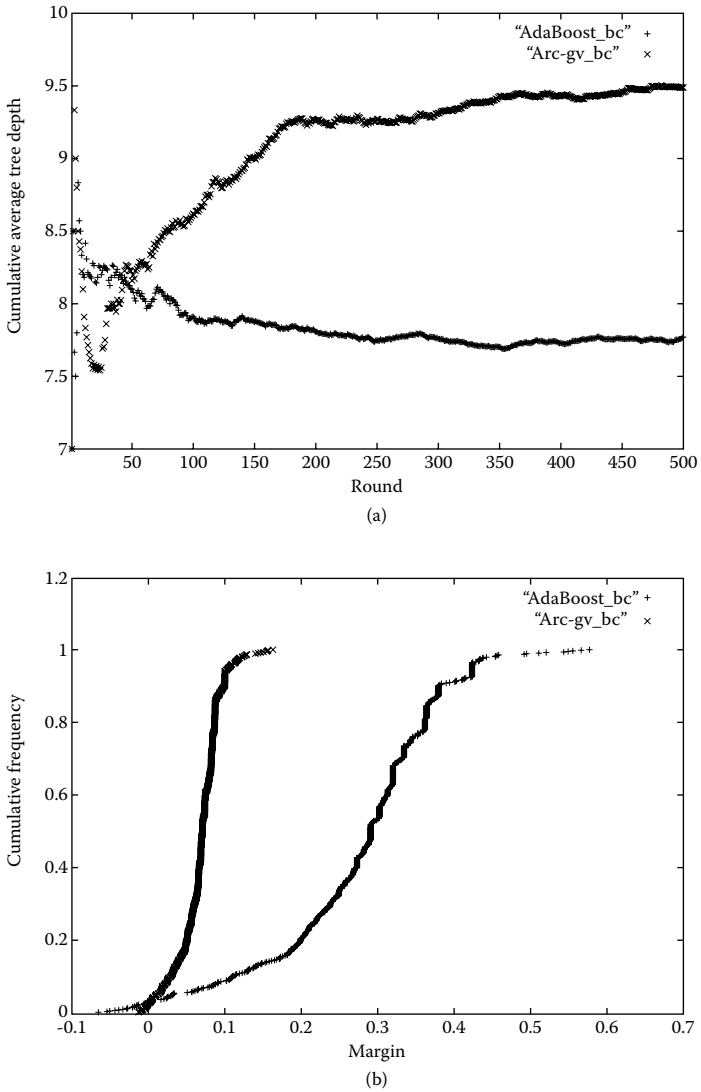


Figure 7.11 Tree depth (top) and margin distribution (bottom) of AdaBoost against arc-gv on the breast cancer data set [20].

This modification is derived from the minimization of multiclass exponential loss. It was proved that, similar to the case of binary classification, optimizing the multiclass exponential loss approaches to the optimal Bayesian error, that is,

$$sign[h^*(\boldsymbol{x})] = \arg\max_{y \in \mathcal{Y}} P(y|\boldsymbol{x})$$

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

A popular solution to multiclass classification problem is to decompose the task into multiple binary classification problems. Direct and popular decompositions include *one-vs-rest* and *one-vs-one*. One-vs-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-vs-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 [23] follows the one-vs-rest approach. After training $|\mathcal{Y}|$ number of (binary) AdaBoost classifiers, the real-value output $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$ of each AdaBoost is used instead of the crisp classification to find the most probable class, that is,

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

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

AdaBoost.M2 [9] follows the one-vs-one approach, which minimizes a pseudo-loss. This algorithm is later generalized as AdaBoost.MR [23] 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-vs-one decomposition can also be aggregated by voting or pairwise coupling [13].

Error correcting output codes (ECOCs) [6] can also be used to decompose a multiclass classification problem into a series of binary classification problems. For example, Figure 7.12a shows output codes for four classes using five classifiers. Each classifier is trained to discriminate the $+1$ and -1 classes in the corresponding column. For a test instance, by concatenating the classifications output by the five classifiers, a code vector of predictions is obtained. This vector will be compared with the code vector of the classes (every row in Figure 7.12(a) using Hamming distance, and the class with the shortest distance is deemed the final prediction. According to information theory, when the binary classifiers are independent, the larger the minimum Hamming distance within the code vectors, the smaller the 0/1-loss. Later, a unified framework was proposed for multiclass decomposition approaches [1]. Figure 7.12(b) shows the output codes for one-vs-rest decomposition and Figure 7.12(c) shows the output codes for one-vs-one decomposition, where zeros mean that the classifiers should ignore the instances of those classes.

	H_1	H_2	H_3	H_4	H_5
	↓	↓	↓	↓	↓
y_1 =	+1	−1	+1	−1	+1
y_2 =	+1	+1	−1	−1	−1
y_3 =	−1	−1	+1	−1	−1
y_4 =	−1	+1	−1	+1	+1

(a) Original code

	H_1	H_2	H_3	H_4
	↓	↓	↓	↓
y_1 =	+1	−1	−1	−1
y_2 =	−1	+1	−1	−1
y_3 =	−1	−1	+1	−1
y_4 =	−1	−1	−1	+1

(b) One-vs-rest code

	H_1	H_2	H_3	H_4	H_5	H_6
	↓	↓	↓	↓	↓	↓
y_1 =	+1	+1	+1	0	0	0
y_2 =	−1	0	0	+1	+1	0
y_3 =	0	−1	0	−1	0	+1
y_4 =	0	0	−1	0	−1	−1

(c) One-vs-one code

Figure 7.12 ECOC output codes.

7.5.3 Other Advanced Topics

Comprehensibility, that is, understandability of the learned model to user, is desired in many real applications. Similar to other ensemble methods, a serious deficiency of AdaBoost and its variants is the lack of comprehensibility. Even when the base learners are comprehensible models such as small decision trees, the combination of them will lead to a black-box model. Improving the comprehensibility of ensemble methods is an important yet largely understudied direction [33].

In most ensemble methods, all the generated base learners are used in the ensemble. However, it has been proved that stronger ensembles with smaller sizes can be obtained through *selective ensemble*, that is, ensembling some instead of all the available base learners [34]. This finding is different from previous results which suggest that ensemble pruning may sacrifice the generalization ability [17,24], and therefore provides support for better selective ensemble or ensemble pruning methods [18,31].

In many applications, training examples of one class are far more than other classes. Learning algorithms that do not consider class imbalance tend to be overwhelmed by the majority class; however, the primary interest is often on the minority class. Many variants of AdaBoost have been developed for class-imbalance learning [5,14,19,26]. Moreover, a recent study [16] suggests that the performance of AdaBoost could be used as a clue to judge whether a task suffers from class imbalance or not, based on which new powerful algorithms may be designed.

As mentioned before, in addition to the 0/1-loss, boosting can also work with other kinds of loss functions. For example, by considering the ranking loss, RankBoost [8] and AdaRank [30] have been developed for information retrieval tasks.

7.6 Software Implementations

As an off-the-shelf machine learning technique, AdaBoost and its variants have easily accessible codes in Java, MATLAB®, R, and C++.

Java implementations can be found in Weka,[3] one of the most famous open-source packages for machine learning and data mining. Weka includes AdaBoost.M1 algorithm [9], which provides options to choose the base learning algorithms, set the number of base learners, and switch between reweighting and resampling mechanisms. Weka also includes other boosting algorithms, such as LogitBoost [11], MultiBoosting [29], and so on.

MATLAB implementation can be found in Spider.[4] R implementation can be found in R-Project.[5] C++ implementation can be found in Sourceforge.[6] There are also many other implementations that can be found on the Internet.

[3] `http://www.cs.waikato.ac.nz/ml/weka/.`
[4] `http://www.kyb.mpg.de/bs/people/spider/.`
[5] `http://cran.r-project.org/web/packages/.`
[6] `http://sourceforge.net/projects/multiboost.`

7.7 Exercises

1. What is the basic idea of Boosting?
2. In Figure 7.2, why should it break when $\epsilon_t \geq 0.5$?
3. Given a training set

$$\left\{ \begin{array}{c} (\boldsymbol{x}_1 = (+1, 0), y_1 = +1) \\ (\boldsymbol{x}_2 = (0, +1), y_2 = +1) \\ (\boldsymbol{x}_3 = (-1, 0), y_3 = +1) \\ (\boldsymbol{x}_4 = (0, -1), y_4 = +1) \\ (\boldsymbol{x}_5 = (0, 0), y_5 = -1) \end{array} \right\}$$

   is there any linear classifier that can reach zero training error? Why/why not?
4. Given the above training set, show that AdaBoost can reach zero training error by using five linear base classifiers from the following pool.

$$\begin{array}{ll} h_1(\boldsymbol{x}) = 2\boldsymbol{I}[x_1 > 0.5] - 1 & h_2(\boldsymbol{x}) = 2\boldsymbol{I}[x_1 < 0.5] - 1 \\ h_3(\boldsymbol{x}) = 2\boldsymbol{I}[x_1 > -0.5] - 1 & h_4(\boldsymbol{x}) = 2\boldsymbol{I}[x_1 < -0.5] - 1 \\ h_5(\boldsymbol{x}) = 2\boldsymbol{I}[x_2 > 0.5] - 1 & h_6(\boldsymbol{x}) = 2\boldsymbol{I}[x_2 < 0.5] - 1 \\ h_7(\boldsymbol{x}) = 2\boldsymbol{I}[x_2 > -0.5] - 1 & h_8(\boldsymbol{x}) = 2\boldsymbol{I}[x_2 < -0.5] - 1 \\ h_9(\boldsymbol{x}) = +1 & h_{10}(\boldsymbol{x}) = -1 \end{array}$$

5. In the above exercise, will AdaBoost reach nonzero training error for any $T \geq 5$? T is the number of base classifiers.
6. The *nearest neighbor classifier* classifies an instance by assigning it with the label of its nearest training example. Can AdaBoost boost the performance of such classifier? Why/why not?
7. Plot the following functions in a graph within range $z \in [-2, 2]$, and observe their difference.

$$\begin{array}{ll} l_1(z) = \begin{cases} 0, & z \geq 0 \\ 1, & z < 0 \end{cases} & l_2(z) = \begin{cases} 0, & z \geq 1 \\ 1 - z, & z < 1 \end{cases} \\ l_3(z) = (z-1)^2 & l_4(z) = e^{-z} \end{array}$$

   Note that, when $z = yf(\boldsymbol{x})$, l_1, l_2, l_3, and l_4 are functions of 0/1-loss, hinge loss (used by support vector machines), square loss (used by least square regression), and exponential loss (the loss function used by AdaBoost), respectively.
8. Show that the l_2, l_3, and l_4 functions in the above exercise are all convex (l is convex if $\forall z_1, z_2 : l(z_1 + z_2) \geq (l(z_1) + l(z_2))$). Considering a binary classification task $z = yf(\boldsymbol{x})$ where $y = \{-1, +1\}$, find that function to which the optimal solution is the Bayesian optimal solution.
9. Can AdaBoost be extended to solve regression problems? If your answer is yes, how? If your answer is no, why?

10. Run experiments to compare AdaBoost using reweighting and AdaBoost using resampling. You can use Weka implementation and data sets from UCI Machine Learning Repository.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[2] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[3] L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley, 1996.

[4] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517, 1999.

[5] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 107–119, Cavtat-Dubrovnik, Croatia, 2003.

[6] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[7] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[8] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–963, 2003.

[9] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[10] Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting (with discussions). *The Annals of Statistics*, 28(2):337–407, 2000.

[12] S. German, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[13] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26(2):451–471, 1998.

[14] M. V. Joshi, R. C. Agarwal, and V. Kumar. Predicting rare classes: Can boosting make any weak learner strong? In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 297–306, Edmonton, Canada, 2002.

[15] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 433–444, Seattle, WA, 1989.

[16] X.-Y. Liu, J.-X. Wu, and Z.-H. Zhou. Exploratory under-sampling for class-imbalance learning. *IEEE Transactions on Systems, Man and Cybernetics—Part B*, 2009.

[17] D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Proceedings of the 14th International Conference on Machine Learning*, pages 211–218, Nashville, TN, 1997.

[18] G. Martínez-Muñoz and A. Suárez. Pruning in ordered bagging ensembles. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 609–616, Pittsburgh, PA, 2006.

[19] H. Masnadi-Shirazi and N. Vasconcelos. Asymmetric boosting. In *Proceedings of the 24th International Conference on Machine Learning*, pages 609–619, Corvallis, OR, 2007.

[20] L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 753–760, Pittsburgh, PA, 2006.

[21] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[22] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[23] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[24] C. Tamon and J. Xiang. On the boosting pruning problem. In *Proceedings of the 11th European Conference on Machine Learning*, pages 404–412, Barcelona, Spain, 2000.

[25] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[26] P. Viola and M. Jones. Fast and robust classification using asymmetric AdaBoost and a detector cascade. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1311–1318. MIT Press, Cambridge, MA, 2002.

[27] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[28] L. Wang, M. Sugiyama, C. Yang, Z.-H. Zhou, and J. Feng. On the margin explanation of boosting algorithm. In *Proceedings of the 21st Annual Conference on Learning Theory*, pages 479–490, Helsinki, Finland, 2008.

[29] G. I. Webb. MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[30] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398, Amsterdam, The Netherlands, 2007.

[31] Y. Zhang, S. Burer, and W. N. Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7:1315–1338, 2006.

[32] Z.-H. Zhou. Ensemble learning. In S. Z. Li, editor, *Encyclopedia of Biometrics*. Springer, Berlin, 2008.

[33] Z.-H. Zhou, Y. Jiang, and S.-F. Chen. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 16(1):3–15, 2003.

[34] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[35] J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class AdaBoost. Technical report, Department of Statistics, University of Michigan, Ann Arbor, 2006.