CHAPTER

30 ENSEMBLE LEARNING

CHAPTER CONTENTS

Ensemble learning is a framework of combining multiple *weak* learning algorithms to produce a *strong* learning algorithm. In this chapter, two types of ensemble learning approaches, *bagging* and *boosting*, are introduced (Fig. 30.1). Although the idea of ensemble learning can be applied to both regression and classification, only classification is considered in this chapter.

30.1 DECISION STUMP CLASSIFIER

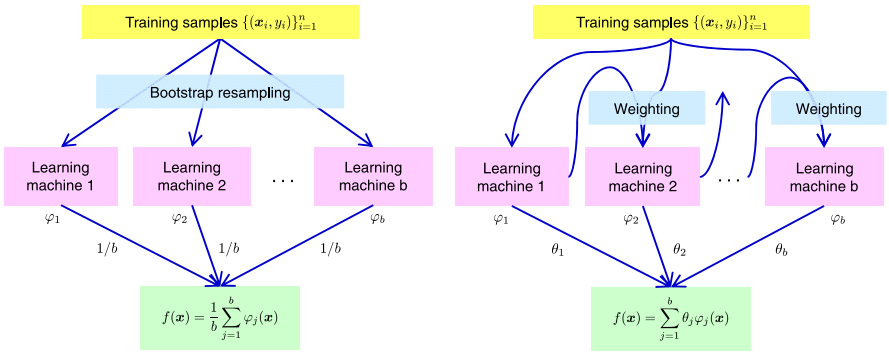
As an example of a weak learning algorithm, let us consider a *decision stump* classifier, which is a depth-one version of *decision trees*. More specifically, a decision stump classifier randomly chooses one of the elements in the d-dimensional input vector $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})^\top$ and classification is performed by thresholding the chosen element. This means that the decision boundary is parallel to one of the coordinate axes (Fig. 30.2).

The decision stump may be a poor classifier in terms of the classification accuracy because of its low degree of freedom. Nevertheless, at least, it has advantage in computation costs because there only exist $n + 1$ solutions for n training samples. Indeed, the global optimal solution can be easily obtained by just sorting the n training samples along the chosen axis and find the best interval that minimizes the classification error.

A MATLAB code for decision stump classification is provided in Fig. 30.3, and its behavior is illustrated in Fig. 30.4. This shows that, as expected, decision stump classification performs poorly.

In the rest of this chapter, ensemble learning methods for improving the performance of this poor decision stump classifier are introduced.

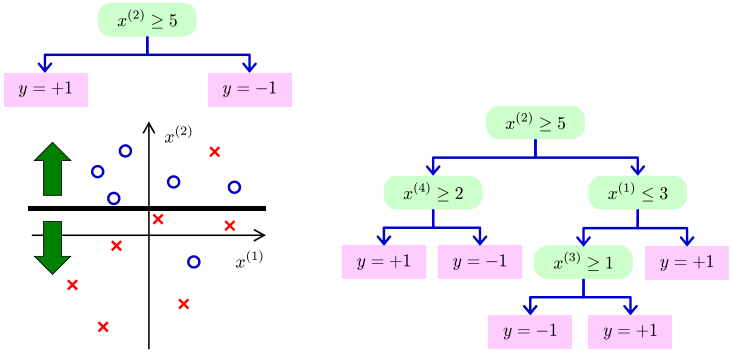
An Introduction to Statistical Machine Learning. DOI: 10.1016/B978-0-12-802121-7.00041-8



(a) Bagging (b) Boosting

FIGURE 30.1

Ensemble learning. Bagging trains weak learners in parallel, while boosting sequentially trains weak learners.



(a) Decision stump (b) Decision tree

FIGURE 30.2

Decision stump and decision tree classifiers. A decision stump is a depth-one version of a decision tree.

30.2 BAGGING

The name, *bagging*, stems from *bootstrap aggregation* [21]. The *bootstrap* method is a resampling technique to generate slightly different data sets from the original training data set (see Section 9.3.2), and bagging aggregates many classifiers trained with slightly different data sets (Fig. 30.1(a)). Since slightly different data sets give slightly different classifiers, averaging them is expected to give a more stable classifier. The algorithm of bagging is described in Fig. 30.5.

```
x=randn(50,2); y=2*(x(:,1)>x(:,2))-1;
X0=linspace(-3,3,50); [X(:,:,1) X(:,:,2)]=meshgrid(X0);

d=ceil(2*rand); [xs,xi]=sort(x(:,d));
el=cumsum(y(xi)); eu=cumsum(y(xi(end:-1:1)));
e=eu(end-1:-1:1)-el(1:end-1); [em,ei]=max(abs(e));
c=mean(xs(ei:ei+1)); s=sign(e(ei)); Y=sign(s*(X(:,:,d)-c));

figure(1); clf; hold on; axis([-3 3 -3 3]);
colormap([1 0.7 1; 0.7 1 1]); contourf(X0,X0,Y);
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
```

FIGURE 30.3

MATLAB code for decision stump classification.

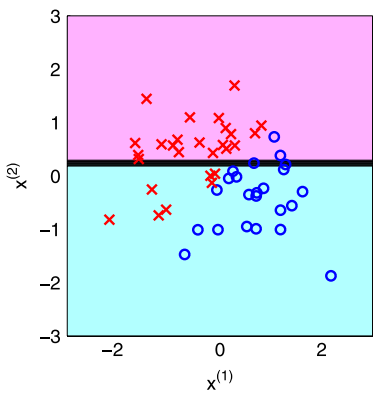


FIGURE 30.4

Example of decision stump classification.

A MATLAB code of bagging for decision stumps is provided in Fig. 30.6, and its behavior is illustrated in Fig. 30.7. This shows that ensemble learning by bagging improves the performance of a decision stump significantly. Note that each bootstrap training procedure is independent of each other, and thus bagging can be computed efficiently in parallel computing environments.

Bagging applied to decision trees is called the *random forest* [22], which is known to be a highly practical algorithm.

1. For $j = 1, \ldots, b$
   (a) Randomly choose n samples from $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ *with replacement* (see Fig. 3.3).
   (b) Train a classifier φ_j with the randomly resampled data set.
2. Output the average of $\{\varphi_j\}_{j=1}^{b}$ as the final solution f:

$$f(\boldsymbol{x}) \longleftarrow \frac{1}{b}\sum_{j=1}^{b}\varphi_j(\boldsymbol{x}).$$

FIGURE 30.5

Algorithm of bagging.

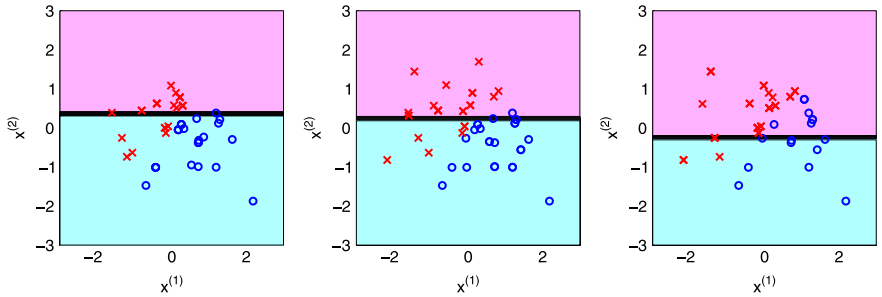
```
n=50; x=randn(n,2); y=2*(x(:,1)>x(:,2))-1;
b=5000; a=50; Y=zeros(a,a);
X0=linspace(-3,3,a); [X(:,:,1) X(:,:,2)]=meshgrid(X0);

for j=1:b
  db=ceil(2*rand); r=ceil(n*rand(n,1));
  xb=x(r,:); yb=y(r); [xs,xi]=sort(xb(:,db));
  el=cumsum(yb(xi)); eu=cumsum(yb(xi(end:-1:1)));
  e=eu(end-1:-1:1)-el(1:end-1);
  [em,ei]=max(abs(e)); c=mean(xs(ei:ei+1));
  s=sign(e(ei)); Y=Y+sign(s*(X(:,:,db)-c))/b;
end

figure(1); clf; hold on; axis([-3 3 -3 3]);
colormap([1 0.7 1; 0.7 1 1]); contourf(X0,X0,sign(Y));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
```

FIGURE 30.6

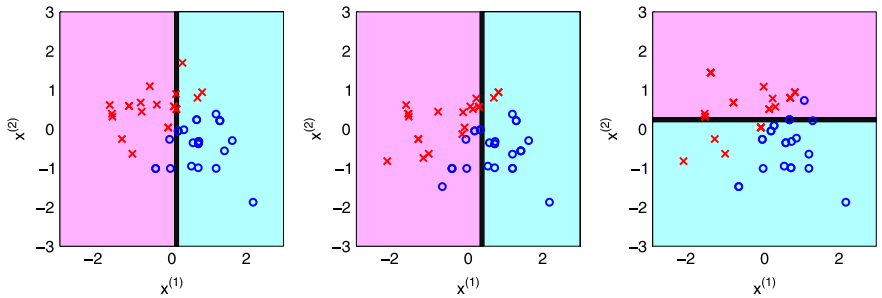
MATLAB code of bagging for decision stumps.

30.3 BOOSTING

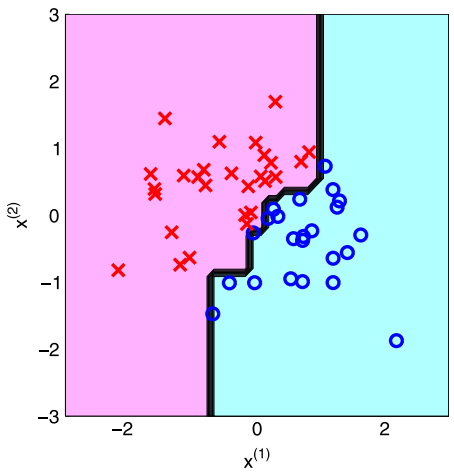
While bagging trained multiple weak learning machines in parallel, *boosting* [87] trains them in a sequential manner (Fig. 30.1(b)). In this section, a fundamental algorithm of boosting is introduced.



(a) $j = 1$ (b) $j = 2$ (c) $j = 3$



(d) $j = 4$ (e) $j = 5$ (f) $j = 6$



(g) Average of $b = 5000$ decision stumps

FIGURE 30.7

Example of bagging for decision stumps.

30.3.1 ADABOOST

In boosting, a weak classifier is first trained with training samples as usual:

$$\{(\boldsymbol{x}_i, y_i) \mid \boldsymbol{x}_i \in \mathbb{R}^d,\ y_i \in \{+1, -1\}\}_{i=1}^n.$$

Because the weak classifier is less powerful, perhaps all training samples cannot be correctly classified. The basic idea of boosting is to assign large weight to the (difficult) training samples that were not correctly classified by the first classifier and train the second classifier with the weights. Thanks to this weighting scheme, the second classifier trained in this way would correctly classify some of the difficult training samples.

Since difficult training samples tend to have large weights, repeating this weighted learning procedure would lead to a classifier that can correctly classify the most difficult training samples. On the other hand, during this iterative procedure, easy training samples have relatively small weights and thus the final classifier may incorrectly classify such easy training samples. For this reason, boosting does not only use the final classifier but also considers weighted voting of all classifiers obtained through the iterative learning procedure. There are various different ways to perform weighted voting, and *adaboost* is one of the popular approaches, which is summarized in Fig. 30.8.

The update formula for sample weights $\{w_i\}_{i=1}^n$ in Fig. 30.8 can be expressed as

$$w_i \longleftarrow w_i \exp\Big(-\theta_j \varphi_j(\boldsymbol{x}_i) y_i\Big),$$

followed by normalization to satisfy $\sum_{i=1}^n w_i = 1$. This implies that adaboost decreases the sample weight if margin $m_i = \varphi_j(\boldsymbol{x}_i) y_i$ is positive (i.e. the ith training sample is correctly classified) and increases the sample weight if margin m_i is negative. The derivation of this update formula will be explained in Section 30.3.2.

The confidence θ_j of weak classifier φ_j in Fig. 30.8,

$$\theta_j = \frac{1}{2} \log \frac{1 - R(\varphi_j)}{R(\varphi_j)},$$

takes a large/small value if the weighted misclassification rate $R(\varphi)$ is small/large (Fig. 30.9). The derivation of this formula will also be explained in Section 30.3.2.

A MATLAB code of adaboost for decision stumps is provided in Fig. 30.10, and its behavior is illustrated in Fig. 30.11. This shows that ensemble learning by boosting gives a strong classifier.

30.3.2 LOSS MINIMIZATION VIEW

The adaboost algorithm introduced above was derived as an ensemble learning method, which is quite different from the LS formulation explained in Chapter 26. However, adaboost can actually be interpreted as an extension of the LS method, and this interpretation allows us to derive, e.g. robust and probabilistic variations of adaboost.

1. Initialize sample weights $\{w_i\}_{i=1}^n$ for $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ to be uniform and the final strong classifier f to be zero:

$$w_1 = \cdots = w_n = 1/n \quad \text{and} \quad f \longleftarrow 0.$$

2. For $j = 1, \ldots, b$

(a) Train a weak classifier φ_j with the current sample weights $\{w_i\}_{i=1}^n$ to minimize the weighted misclassification rate $R(\varphi)$:

$$\varphi_j = \operatorname*{argmin}_{\varphi} R(\varphi), \quad \text{where} \quad R(\varphi) = \sum_{i=1}^{n} \frac{w_i}{2}\Big(1 - \varphi(\boldsymbol{x}_i) y_i\Big).$$

(b) Set the confidence θ_j of weak classifier φ_j at

$$\theta_j = \frac{1}{2} \log \frac{1 - R(\varphi_j)}{R(\varphi_j)}.$$

(c) Update the strong classifier f as

$$f \longleftarrow f + \theta_j \varphi_j.$$

(d) Update the sample weights $\{w_i\}_{i=1}^n$ as

$$w_i \longleftarrow \frac{\exp\Big(-f(\boldsymbol{x}_i) y_i\Big)}{\sum_{i'=1}^n \exp\Big(-f(\boldsymbol{x}_{i'}) y_{i'}\Big)}, \quad \forall i = 1, \ldots, n.$$

FIGURE 30.8

Algorithm of adaboost.

As a surrogate loss to the 0/1-loss (see Section 26.2), let us consider the *exponential loss* (Fig. 30.12):

$$\exp(-m),$$

where $m = f_{\boldsymbol{\theta}}(\boldsymbol{x}) y$ is the margin. For the linear-in-parameter model,

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{b} \theta_j \varphi_j(\boldsymbol{x}),$$

where $\{\varphi_j(\boldsymbol{x})\}_{j=1}^b$ are binary-valued basis functions that only takes either -1 or $+1$, let us consider exponential loss minimization:

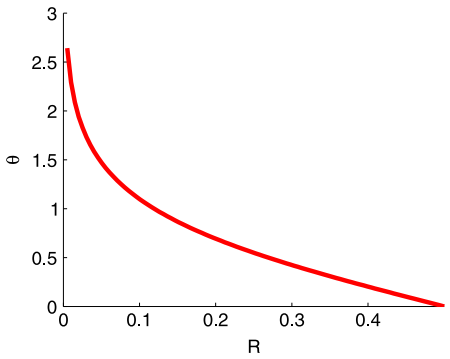


FIGURE 30.9

Confidence of classifier in adaboost. The confidence of classifier φ, denoted by θ, is determined based on the weighted misclassification rate R.

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \exp\Big(- f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) y_i \Big). \tag{30.1}$$

Suppose that not only the parameters $\{\theta_j\}_{j=1}^{b}$ but also the basis functions $\{\varphi_j\}_{j=1}^{b}$ can be learned in a sequential manner one by one. Let $\widetilde{f}$ be the learned function obtained so far and let θ and φ be the next parameter and basis function to be learned, respectively. Then Eq. (30.1) yields weighted exponential loss minimization:

$$\min_{\theta,\varphi} \sum_{i=1}^{n} \exp\Big(- \big\{ \widetilde{f}(\boldsymbol{x}_i) + \theta\varphi(\boldsymbol{x}_i) \big\} y_i \Big) = \min_{\theta,\varphi} \sum_{i=1}^{n} \widetilde{w}_i \exp\Big(- \theta\varphi(\boldsymbol{x}_i) y_i \Big),$$

where the weight $\widetilde{w}_i$ is given by

$$\widetilde{w}_i = \exp\Big(- \widetilde{f}(\boldsymbol{x}_i) y_i \Big).$$

For the sake of simplicity, $\theta \geq 0$ is assumed below (when $\theta < 0$, the sign of φ is flipped to satisfy $\theta \geq 0$). Then the above sequential weighted exponential loss minimization can be rewritten as

$$\begin{aligned}
&\sum_{i=1}^{n} \widetilde{w}_i \exp\Big(- \theta\varphi(\boldsymbol{x}_i) y_i \Big) \\
&\quad = \exp(-\theta) \sum_{i : y_i = \varphi(\boldsymbol{x}_i)} \widetilde{w}_i + \exp(\theta) \sum_{i : y_i \neq \varphi(\boldsymbol{x}_i)} \widetilde{w}_i \\
&\quad = \big\{ \exp(\theta) - \exp(-\theta) \big\} \sum_{i=1}^{n} \frac{\widetilde{w}_i}{2} \Big(1 - \varphi(\boldsymbol{x}_i) y_i \Big) + \exp(-\theta) \sum_{i=1}^{n} \widetilde{w}_i .
\end{aligned} \tag{30.2}$$

```
n=50; x=randn(n,2); y=2*(x(:,1)>x(:,2))-1; b=5000;
a=50; Y=zeros(a,a); yy=zeros(size(y)); w=ones(n,1)/n;
X0=linspace(-3,3,a); [X(:,:,1) X(:,:,2)]=meshgrid(X0);

for j=1:b
  wy=w.*y; d=ceil(2*rand); [xs,xi]=sort(x(:,d));
  el=cumsum(wy(xi)); eu=cumsum(wy(xi(end:-1:1)));
  e=eu(end-1:-1:1)-el(1:end-1);
  [em,ei]=max(abs(e)); c=mean(xs(ei:ei+1)); s=sign(e(ei));
  yh=sign(s*(x(:,d)-c)); R=w'*(1-yh.*y)/2;
  t=log((1-R)/R)/2; yy=yy+yh*t; w=exp(-yy.*y); w=w/sum(w);
  Y=Y+sign(s*(X(:,:,d)-c))*t;
end

figure(1); clf; hold on; axis([-3 3 -3 3]);
colormap([1 0.7 1; 0.7 1 1]); contourf(X0,X0,sign(Y));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
```

FIGURE 30.10

MATLAB code of adaboost for decision stumps.

Thus, the minimizer $\widehat{\varphi}$ of Eq. (30.2) with respect to φ is given by weighted 0/1-loss minimization:

$$\widehat{\varphi} = \operatorname*{argmin}_{\varphi} \sum_{i=1}^{n} \frac{\widetilde{w}_i}{2} \big(1 - \varphi(\boldsymbol{x}_i) y_i\big).$$

The minimizer $\widehat{\theta}$ of Eq. (30.2) with respect to θ can be obtained by substituting $\widehat{\varphi}$ to φ in Eq. (30.2), differentiating it with respect to θ, and setting it at 0:

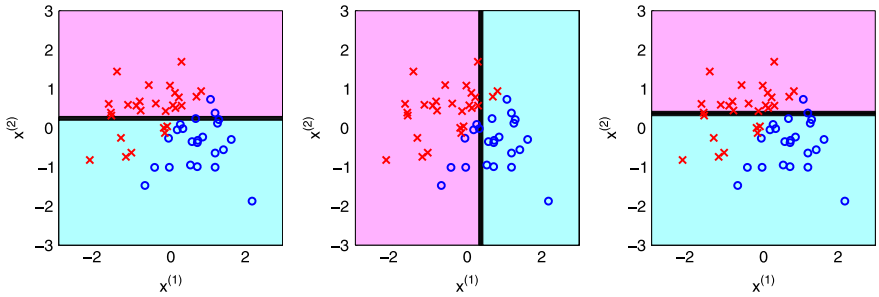
$$\big\{\exp(\theta) + \exp(-\theta)\big\} \sum_{i=1}^{n} \frac{\widetilde{w}_i}{2} \big(1 - \widehat{\varphi}(\boldsymbol{x}_i) y_i\big) - \exp(-\theta) \sum_{i=1}^{n} \widetilde{w}_i = 0.$$

Solving this for θ yields

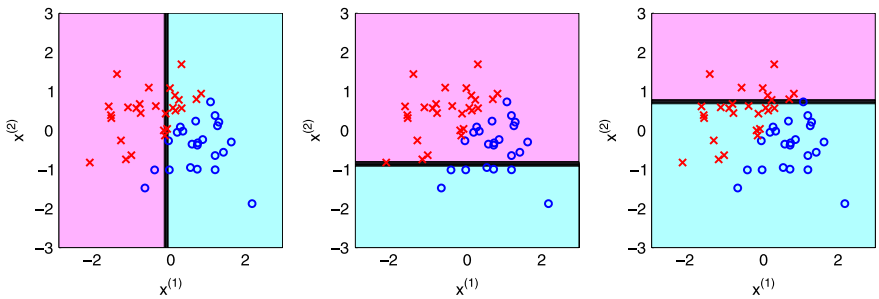
$$\widehat{\theta} = \frac{1}{2} \log \frac{1 - \widehat{R}}{\widehat{R}},$$

where $\widehat{R}$ denotes the weighted 0/1-loss for $\widehat{\varphi}$:

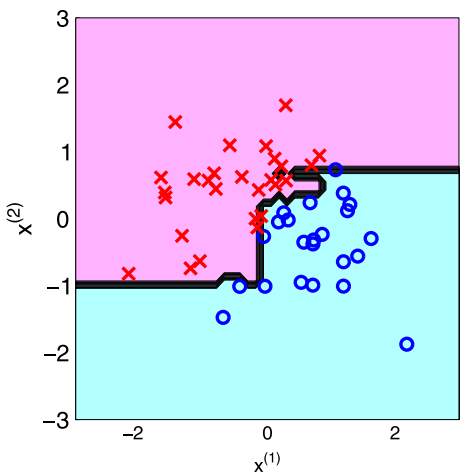
$$\widehat{R} = \left\{ \sum_{i=1}^{n} \frac{\widetilde{w}_i}{2} \big(1 - \widehat{\varphi}(\boldsymbol{x}_i) y_i\big) \right\} \Big/ \left\{ \sum_{i'=1}^{n} \widetilde{w}_{i'} \right\}.$$



(a) $j = 1$ (b) $j = 2$ (c) $j = 3$



(d) $j = 4$ (e) $j = 5$ (f) $j = 6$



(g) Combination of $b = 5000$ decision stumps

FIGURE 30.11

Example of adaboost for decision stumps.

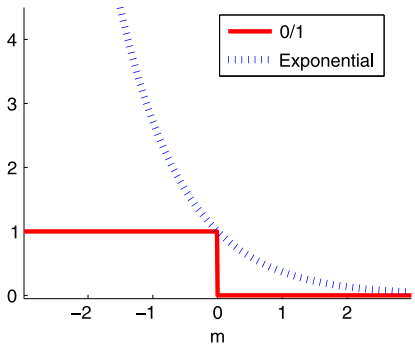


FIGURE 30.12

Exponential loss.

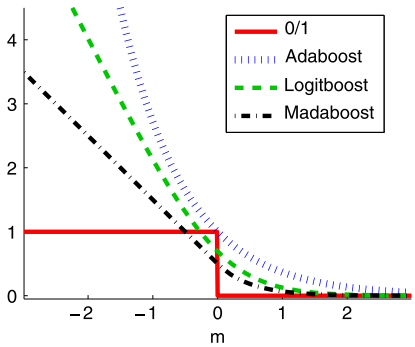


FIGURE 30.13

Loss functions for boosting.

Since $\widehat{\varphi}$ and $\widehat{\theta}$ obtained above are equivalent to the one used in adaboost (see Fig. 30.8), adaboost was shown to be equivalent to sequential weighted exponential loss minimization.

The above equivalence allows us to consider variations of adaboost for other loss functions. For example, a modification of adaboost, called *madaboost* [37], uses

$$\begin{cases} -m + 1/2 & (m \leq 0) \\ \exp(-2m)/2 & (m > 0) \end{cases}$$

as a loss function (Fig. 30.13). This loss function leads to *robust* classification since it increases only linearly. Another popular variation is *logitboost* [43], which uses

$$\log(1 + \exp(-2m))$$

as a loss function (Fig. 30.13). Logitboost corresponds to a boosting version of *logistic regression* introduced in Section 28.1, which allows probabilistic interpretation.

30.4 GENERAL ENSEMBLE LEARNING

Beyond systematic ensemble learning methods such as bagging and boosting, voting by various different learning algorithms is highly promising in practical applications. For example, the winning algorithms in real-world data analysis competitions such as *Netflix Prize*[1] and *KDD Cup 2013*[2] adopt such voting approaches. If additional validation data samples are available, voting weights may be optimized so that the prediction error for the validation samples is minimized.

Thus, beyond improving each learning algorithm, ensemble learning can be regarded as a final means to boost the prediction performance.

[1] http://www.netflixprize.com/.
[2] http://www.kdd.org/kddcup2013/content/kdd-cup-2013-workshop-0.