Introduction to AdaBoost

Balázs Kégl

January 21, 2014

Contents

Notations - Definitions

General

- $\mathcal{X}$: measurable observation space; usually $\mathcal{X} = \mathbb{R}^d$
- X : random observation vector in $\mathcal{X}$
- $\mathbf{x}$: fixed observation vector in $\mathcal{X}$
- d : dimension of the observation space (if finite)
- K : number of classes in multi-class classification
- $\mathcal{Y}$: label space; $\mathcal{Y} = \{-1, 1\}$ in binary classification, $\mathcal{Y} \subseteq \mathbb{R}$ in regression, $\mathcal{Y} = \{1, \ldots, K\}$ in multi-class classification, and $\mathcal{Y} = \{0, 1\}^K$ in multi-class multi-label classification.
- Y : random label in $\mathcal{Y}$
- y : fixed label in $\mathcal{Y}$
- $Z = (X, Y)$
- $\mathcal{D}_n = \big\{(X_1, Y_1), \ldots, (X_n, Y_n)\big\}$ or $\mathcal{D}_n = \big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\big\}$: training sample of size n
- n : training sample size
- m : test sample size
- P : distribution of (X, Y)
- $P^{\otimes n}$: distribution of $(X_1, Y_1), \ldots, (X_n, Y_n)$
- μ : marginal distribution of X
- η : regression function, $\forall \mathbf{x} \in \mathcal{X}, \eta(\mathbf{x}) = \mathbb{E}\{Y \mid X = \mathbf{x}\}$.
- R : risk
- $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$: loss function
- $f : \mathcal{X} \to \mathcal{Y}$: predictor
- $\mathcal{F}$: restricted set of predictors
- Definition: risk of predictor f

$$R(f) = \mathbb{E}\left\{\ell\big(Y, f(X)\big)\right\} = \int_{\mathcal{X} \times \mathcal{Y}} \ell\big(y, f(\mathbf{x})\big)\, dP(\mathbf{x}, y)$$

- f^* : Bayes predictor
- $f_{\mathcal{F}}$: best predictor in the set $\mathcal{F}$
- R^* : Bayes risk
- Definition: empirical risk (training error) of predictor f

$$R_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(Y_i, f(X_i)\big)$$

- $\widehat{f}_n$: estimator based on $\mathcal{D}_n$

  B: For the time being I keep $\mathcal{F}$ and f but we may switch to g and $\mathcal{G}$ all above.

  N: OK

Classification

- $h : \mathcal{X} \to \mathcal{Y}$: base classifier
- $\mathcal{H}$: set of base classifiers
- $\mathcal{Z}$: discriminant space, $\mathcal{Z} = \mathbb{R}$ in binary classification, $\mathcal{Z} = \mathbb{R}^K$ in multi-class (and multi-class multi-label) classification.
- $f : \mathcal{X} \to \mathcal{Z}$: discriminant function
- $\mathcal{F}$: set of discriminant functions
- g_f : classifier obtained from the discriminant function f. In binary classification $g_f = 2\mathbb{I}\{f > 0\} - 1$, in multi-class classification $g_f = \arg\max_{k\in\{1,\dots,K\}} f_k$
- $\mathcal{F}_N$: set of finite convex combinations of N elements of $\mathcal{H}$
- $\mathcal{F}_\lambda$: set of linear combinations of elements of $\mathcal{H}$ such that the 1-norm of the coefficients is equal to λ.
- $\mathcal{F}_{N,\lambda}$: set of linear combinations of N elements of $\mathcal{H}$ such that the 1-norm of the coefficients is equal to λ.
- α_j : real valued coefficients
- $f = \sum_{j\geq 1} \alpha_j h_j$: element of $\mathcal{F}$
- $\alpha = (\alpha_1, \dots)$: coefficient vector
- $\|\alpha\|_1 = \sum_{j\geq 1} \alpha_j$: 1-norm of the coefficient vector
- $\widetilde{\alpha}_j = \frac{\alpha_j}{\|\alpha\|_1}$: normalized coefficients
- $\widetilde{\alpha} = (\widetilde{\alpha}_1, \dots)$: normalized coefficient vector (discrete probability distribution over $\mathcal{H}$)
- $\widetilde{f} = \sum_{j\geq 1} \widetilde{\alpha}_j h_j \in \mathcal{F}_1$: normalized discriminant function
- $g : \mathcal{X} \to \{-1, 1\}$: final (binary) classifier
- $\mathcal{G}$: set of classifiers
- V : VC dimension

- $\mathbf{w} = (w_1, \ldots, w_n)$: distribution over $\mathcal{D}_n$
- $g_f = 2\mathbb{I}\{f > 0\} - 1$: classifier obtained from the real-valued predictor f
- Definition: risk of classifier g

$$L(g) = \mathbb{P}\{Y \neq g(X)\} = \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{I}\{y \neq g(\mathbf{x})\}\, dP(\mathbf{x}, y)$$

- g^* : Bayes classifier
- $g_{\mathcal{G}}$: best classifier in the restricted set $\mathcal{G}$
- Definition: empirical risk of classifier g

$$L_n(g) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{Y_i \neq g(X_i)\}$$

- $\widehat{g}_n$: estimator based on $\mathcal{D}_n$
- φ : (surrogate) convex cost function
- Definition: convex risk of predictor f

$$A(f) = \mathbb{E}\left\{\varphi\big(-Y f(X)\big)\right\}$$

- Definition: empirical convex risk of predictor f

$$A_n(f) = \frac{1}{n}\sum_{i=1}^{n} \varphi\big(-Y_i f(X_i)\big)$$

B: I came back on this issue to an earlier proposition. I know it's not perfect, but I don't know what to do. Clearly, in the algorithmic section the data points will be given, so the margin varies with the discriminant function (so f is the more natural variable), while in the statistical section the function is given, and the natural variable is the data point X.

- $\rho_Z(f) = f(X)Y$: margin of a discriminant function f on data point (X, Y) (multi-class, regression: later)
- $\widetilde{\rho}_Z(f) = \widetilde{f}(X)Y$: normalized margin of a discriminant function f on data point (X, Y)
- ρ_i ($\widetilde{\rho}_i$) : margin (normalized margin) of a discriminant function (omitted in the notation) on the training data point (X_i, Y_i) or $(\mathbf{x}_i, y_i)$.
- $\rho_i(\alpha)$ ($\widetilde{\rho}_i(\alpha)$) : margin (normalized margin) of the discriminant function $\sum_{j\geq 1} \alpha_j h_j$ on the training point (X_i, Y_i) or $(\mathbf{x}_i, y_i)$.
- $\rho(\alpha)$ ($\widetilde{\rho}(\alpha)$) : margin (normalized margin) vector over the data set $\mathcal{D}_n$.
- Definition: marginal empirical risk of a discriminant function f

$$L_n^{(\rho)}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{f(X_i)Y_i < \rho\} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\rho_i < \rho\}$$

Algorithm

- T : number of iterations in the algorithm
- t : iteration index
- $\mathbf{w}^{(t)}$: distribution over $\mathcal{D}_n$ in the tth iteration (computed before the tth iteration)
- $h^{(t)}$: base classifier selected in the tth iteration
- $\varepsilon^{(t)} = \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}$: weighted error of base classifier in the tth iteration
- $\gamma^{(t)} = 1 - 2\varepsilon^{(t)}$: edge of $h^{(t)}$
- $\alpha^{(t)}$: coefficient of $h^{(t)}$
- $f^{(t)} = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}$: aggregate discriminant function, output of ADABOOST
- $\rho_i^{(t)} = \rho_{f^{(t)}}(Z_i)$: margin of the ith training point after the tth iteration

Chapter 1

Introduction

Definitions: observations, features (aka. attribute, coordinate), labels, classifier, discriminant function, margin, loss, risk, empirical loss, empirical risk, empirical risk minimization, margin risk functions, hyper-parameters, regularization (give just hints, no deep explanation how they are related).

For the time being I'm defining everything where I'm using it. For pedagogical reasons we could keep this policy: give basic definitions in this chapter, so if somebody wants to look them up, everything can be found here, but also define them later when they are first used in the boosting context.

Notations: bold for vectors: $\mathbf{x}$; same letter but italic for the elements: x_i; capital letter for random variables (vector or scalar): X, Y; capital bold for matrices, random or not: $\mathbf{X}$. Same letter for indices over the same variable: i over data points, j over base classifiers, t over iterations, ℓ over classes (multiclass).

Due to its simplicity, ADABOOST is also a very good algorithm to introduce machine learning. It can be taught after a first year programming and algorithms class. The key elements of binary AdaBoosting decision stumps is completely elementary. At the same time, it's state-of-the-art on benchmark tests. Implies some choices, e.g., we give the pseudocode of decision stumps in the second chapter so the whole algorithm can be programmed from scratch.

Our main audiance is a novice, but we hope to give some novelty (or at least synthetize) for the ML expert (who is not necessarily a boosting expert).

Although several questions remain unanswered (no foolproof explanation why it "works"), it is a very good algo in practice, so deserves a monographe.

1.1 Bibliographical notes

Bishop, Duda-Hart, Devroye-Györfi-Lugosi

Chapter 2

The basic algorithm: binary ADABOOST

In this chapter we describe the basic ADABOOST algorithm used to learn binary (two-class) classifiers. Algorithmic issues, algorithmic convergence results. Section on base classifiers. Simple example.

2.1 Basics

data, error, margin, error minimization

2.2 The binary ADABOOST algorithm

ADABOOST is an *ensemble* (or *meta-learning*) method that constructs a classifier in an iterative fashion. In each iteration, it calls a simple learning algorithm (called the *base learner*) that returns a classifier, and assigns a weight coefficient to it. The final classification will be decided by a weighted "vote" of the base classifiers. The smaller the error of the base classifier, the larger is its weight in the final vote. The base classifiers have to be only slightly better than a random guess (from where their alternative name *weak classifier* derives), which gives great flexibility to the design of the base classifier (or feature) set.

For the formal description of ADABOOST, let the training set be $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. The algorithm runs for T iterations. T is the only pre-specified hyper-parameter of ADABOOST that can be set by, for example, cross-validation. In each iteration $t = 1, \ldots, T$, we choose a base classifier $h^{(t)}$ from a set $\mathcal{H}$ of classifiers and set its coefficient $\alpha^{(t)}$. In the simplest version of the algorithm, $\mathcal{H}$ is a finite set of binary classifiers of the form $h : \mathbb{R}^d \to \{-1, 1\}$, and the base learner executes an exhaustive search over $\mathcal{H}$ in each iteration. The output of ADABOOST is a *discriminant function* constructed as a weighted vote of the base classifiers

$$f^{(T)}(\cdot) \triangleq \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot). \tag{2.1}$$

The sign of $f^{(T)}(\mathbf{x})$ is then used as the final classification of $\mathbf{x}$.

The algorithm maintains a weight distribution $\mathbf{w}^{(t)} = \left(w_1^{(t)}, \ldots, w_n^{(t)}\right)$ over the data points[1]. The weights are initialized uniformly in line 1, and are updated in each iteration in lines 7-10 (Figure 2.1). The goal of the base learner is to minimize the *weighted (base) error*

$$\varepsilon^{(t)} \triangleq \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}.^{2} \tag{2.2}$$

Reference to decision stumps.

[1] To avoid confusion, from now on we will call the base classifiers' weights $\alpha^{(t)}$ *coefficients*, and keep the term *weight* for the weights w_i of the data points.

[2] The indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

The coefficient $\alpha^{(t)}$ of $h^{(t)}$ is set in line 5 to

$$\alpha^{(t)} \triangleq \frac{1}{2}\ln\left(\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}\right). \tag{2.3}$$

ADABOOST$\left(\mathcal{D}_n, \text{BASE}(\cdot,\cdot), T\right)$

1 $\mathbf{w}^{(1)} \leftarrow (1/n, \ldots, 1/n)$ ▷ *initial weights*
2 **for** $t \leftarrow 1$ **to** T
3 $\quad h^{(t)} \leftarrow \text{BASE}\left(\mathcal{D}_n, \mathbf{w}^{(t)}\right)$ ▷ *base classifier*
4 $\quad \varepsilon^{(t)} \leftarrow \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}$ ▷ *weighted error of the base classifier*
5 $\quad \alpha^{(t)} \leftarrow \frac{1}{2}\ln\left(\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}\right)$ ▷ *coefficient of the base classifier*
6 $\quad$ **for** $i \leftarrow 1$ **to** n ▷ *re-weighting the training points*
7 $\quad\quad$ **if** $h^{(t)}(\mathbf{x}_i) \neq y_i$ **then** ▷ *error*
8 $\quad\quad\quad w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\varepsilon^{(t)}}$ ▷ *weight increases*
9 $\quad\quad$ **else** ▷ *correct classification*
10 $\quad\quad\quad w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\left(1-\varepsilon^{(t)}\right)}$ ▷ *weight decreases*
11 **return** $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$ ▷ *weighted "vote" of base classifiers*

Figure 2.1: The pseudocode of ADABOOST. $\mathcal{D}_n = \left\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\right\}$ is the training set, BASE$(\cdot,\cdot)$ is the base learner, and T is the number of iterations.

The weights remain normalized throughout the algorithm, that is, for all t

$$\sum_{i=1}^{n} w_i^{(t)} = 1. \tag{2.4}$$

To see that, first note that the initialization guarantees that $\sum_{i=1}^{n} w_i^{(1)} = 1$. Then, assuming that $\sum_{i=1}^{n} w_i^{(t)} = 1$, we have

$$\begin{aligned}
\sum_{i=1}^{n} w_i^{(t+1)} &= \sum_{i=1}^{n} \frac{w_i^{(t)}}{2\varepsilon^{(t)}} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\} + \sum_{i=1}^{n} \frac{w_i^{(t)}}{2\left(1-\varepsilon^{(t)}\right)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) = y_i\right\} && (2.5)\\
&= \frac{1}{2\varepsilon^{(t)}} \sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\} + \frac{1}{2\left(1-\varepsilon^{(t)}\right)} \sum_{i=1}^{n} w_i^{(t)} \left(1 - \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}\right) && (2.6)\\
&= \frac{1}{2\varepsilon^{(t)}} \varepsilon^{(t)} + \frac{1}{2\left(1-\varepsilon^{(t)}\right)} \left(1-\varepsilon^{(t)}\right) = \frac{1}{2} + \frac{1}{2} = 1. && (2.7)
\end{aligned}$$

In (2.5) we applied the weight update rules in lines 8 and 10, in (2.6) we used the fact that $\mathbb{I}\{A\} = 1 - \mathbb{I}\{\neg A\}$, and in (2.7) we applied the definition of $\varepsilon^{(t)}$ in (2.2) and the inductive assumption of $\sum_{i=1}^{n} w_i^{(t)} = 1$.

Since the weights are normalized, $\varepsilon^{(t)}$ is a real number between 0 and 1. Moreover, if $\mathcal{H}$ is closed under negation,[3] we can guarantee that $\varepsilon^{(t)} \leq \frac{1}{2}$, since otherwise we would flip signs and use $-h^{(t)}$ instead of $h^{(t)}$.

The first implication of $\varepsilon^{(t)} \leq \frac{1}{2}$ is that $\alpha^{(t)}$ is always non-negative. The coefficient monotonically increases as $\varepsilon^{(t)}$ decreases (Figure 2.2), which means that the better the base classifier $h^{(t)}$, the higher is its "vote" in the final classification. As the base error $\varepsilon^{(t)}$ goes to 0, the coefficient $\alpha^{(t)}$ tends to infinity. Formally, a "strong" base classifier with $\varepsilon^{(t)} = 0$ will automatically receive an infinite coefficient, so it will dominate the linear combination (2.1). In principle, since ADABOOST's objective is to minimize the training error, $\varepsilon^{(t)} = 0$ means that the objective has been achieved, so it is normal to terminate the algorithm. In certain variants of ADABOOST it will be useful to be able to continue even if $\varepsilon^{(t)} = 0$; we will discuss how to deal with this problem in Section **??**.

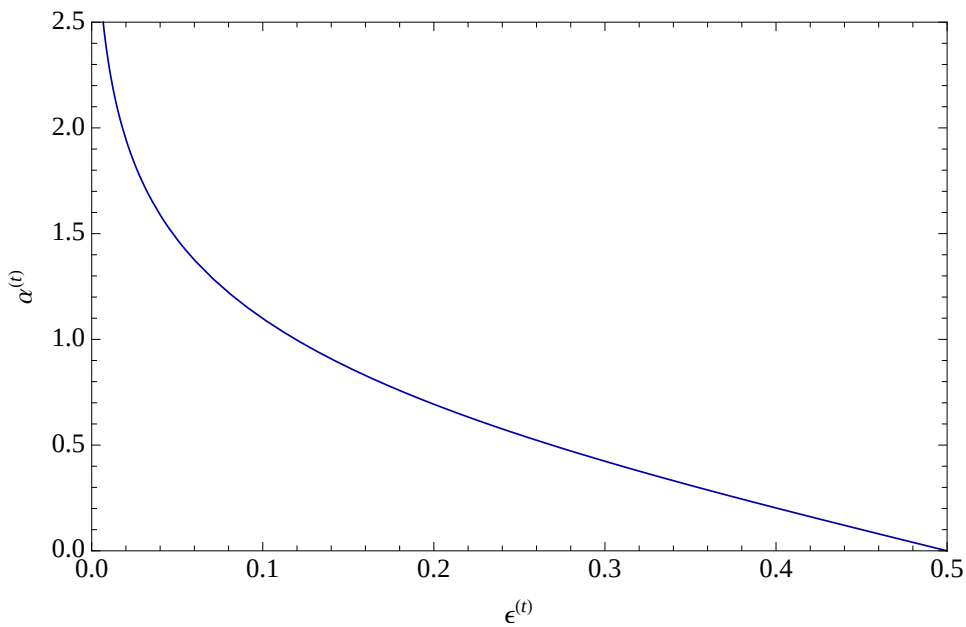


Figure 2.2: The base coefficient $\alpha^{(t)}$ as a function of the base error $\varepsilon^{(t)}$. At $\varepsilon^{(t)} = \frac{1}{2}$, $\alpha^{(t)} = 0$, and $\alpha^{(t)} \to \infty$ as $\varepsilon^{(t)} \to 0$.

The second implication of $\varepsilon^{(t)} \leq \frac{1}{2}$ is that when the weights w_i of training points are updated in lines 7-10, the weights of misclassified points increase (line 8), and the weights of correctly classified points decrease (line 10). As the algorithm progresses, the weights of frequently misclassified points will tend to be large, so base classifiers will concentrate more and more on these "hard" data points.

Finally, by looking at the evolution of the first summand of (2.5), we can also note that the base error $\varepsilon^{(t+1)}$ of the tth base classifier $h^{(t)}$ is $\frac{1}{2}$, so we never select the same base classifier in two consecutive iterations (unless we are in a degenerate case).

[3] That is, if $h \in \mathcal{H}$ then $-h \in \mathcal{H}$. This property is satisfied by most of the practical classifier classes, and we will assume it from now on. If $\mathcal{H}$ is not closed under negation and we cannot find a base classifier with $\varepsilon^{(t)} \leq \frac{1}{2}$, then the algorithm terminates and returns $f^{(t-1)}$. In the degenerate case when $\varepsilon^{(t)} = \frac{1}{2}$, we can continue the loop, but since $\alpha^{(\tau)} = 0$ for all $\tau \geq t$, we should also terminate in practice.

2.3 Basic analysis

First we unify the two cases in the weight update formulae (lines 7-10):

$$
\begin{aligned}
w_i^{(t+1)} &= w_i^{(t)} \times \begin{cases} \frac{1}{2\varepsilon^{(t)}} & \text{if } h^{(t)}(\mathbf{x}_i) \neq y_i \\ \frac{1}{2\left(1-\varepsilon^{(t)}\right)} & \text{if } h^{(t)}(\mathbf{x}_i) = y_i \end{cases} \\
&= w_i^{(t)} \times \begin{cases} \frac{\sqrt{\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}}}}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}} & \text{if } h^{(t)}(\mathbf{x}_i) \neq y_i \\ \frac{\sqrt{\frac{\varepsilon^{(t)}}{1-\varepsilon^{(t)}}}}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}} & \text{if } h^{(t)}(\mathbf{x}_i) = y_i \end{cases} \\
&= w_i^{(t)} \times \begin{cases} \frac{e^{\alpha^{(t)}}}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}} & \text{if } h^{(t)}(\mathbf{x}_i)y_i = -1 \\ \frac{e^{-\alpha^{(t)}}}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}} & \text{if } h^{(t)}(\mathbf{x}_i)y_i = 1 \end{cases} \qquad (2.8) \\
&= w_i^{(t)} \frac{\exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)}{2\sqrt{\varepsilon^{(t)}\left(1-\varepsilon^{(t)}\right)}}, \qquad (2.9)
\end{aligned}
$$

where in (2.8) we use the definition (2.3) of the coefficient $\alpha^{(t)}$ and the fact that both y_i and $h^{(t)}(\mathbf{x}_i)$ are $\{-1,1\}$-valued. Using the unified equation (2.9), the definition (2.1) of the final discriminant function $f^{(T)}$, and the uniform initialization of the weights $w_i^{(1)}$ in line 1, we can express the weights in a non-recursive way as

$$
\begin{aligned}
w_i^{(t+1)} &= w_i^{(t)} \frac{\exp\left(-\alpha^{(t)} h^{(t)}(\mathbf{x}_i) y_i\right)}{2\sqrt{\varepsilon^{(t)}(1-\varepsilon^{(t)})}} = \ldots = \frac{1}{n} \times \frac{\exp\left(-y_i \sum_{\tau=1}^{t} \alpha^{(\tau)} h^{(\tau)}(\mathbf{x}_i)\right)}{\prod_{\tau=1}^{t} 2\sqrt{\varepsilon^{(\tau)}(1-\varepsilon^{(\tau)})}} \\
&= \frac{\exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right)}{n \prod_{\tau=1}^{t} 2\sqrt{\varepsilon^{(\tau)}(1-\varepsilon^{(\tau)})}}. \qquad (2.10)
\end{aligned}
$$

First, this non-recursive formula tells us that the weights are proportional to $\exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right)$. The negative exponent

$$\rho_i^{(t)} \triangleq f^{(t)}(\mathbf{x}_i) y_i$$

is the *unnormalized margin*, defined in general for a discriminant function f and a data point $(\mathbf{x}, y)$ as

$$\rho_{\mathbf{x},y}(f) \triangleq f(\mathbf{x}) y. \qquad (2.11)$$

This quantity will play a crucial role in the margin-based analysis of the generalization error (Section **??**). For now it is clear, that 1) the sign of the margin $\rho_i^{(t)}$ indicates whether $(\mathbf{x}_i, y_i)$ is correctly classified by $f^{(t)}$ or not, and 2) the larger $\rho_i^{(t)}$, the more "confident" the classification of $(\mathbf{x}_i, y_i)$ is, so intuitively it makes sense that the weight $w_i^{(t+1)}$ decreases monotonically with $\rho_i^{(t)}$.

Second, from (2.7) we know that the weights $w_i^{(t+1)}$ sum to 1, so from the non-recursive formula (2.10) it follows that

$$w_i^{(t+1)} = \frac{\exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right)}{\sum_{\iota=1}^{n} \exp\left(-f^{(t)}(\mathbf{x}_\iota) y_\iota\right)}. \qquad (2.12)$$

By comparing (2.10) and (2.12) we automatically obtain the key equation

$$\frac{1}{n}\sum_{i=1}^{n}\exp\left(-f^{(t)}(\mathbf{x}_i)y_i\right)=\prod_{\tau=1}^{t}2\sqrt{\varepsilon^{(\tau)}(1-\varepsilon^{(\tau)})} \tag{2.13}$$

of the convergence theorem:

Theorem 1 (Theorem 6 in [FS97]). Assuming that there exists a positive constant δ for which the base error is upper bounded by

$$\varepsilon^{(t)}\leq\frac{1}{2}-\delta \tag{2.14}$$

(*** maybe $\frac{1-\delta}{2}$ would be better: lower bound on the edge) for all t, the training error $R_n\left(f^{(T)}\right)$ becomes 0 after at most

$$T=\left\lceil\frac{\ln n}{2\delta^2}\right\rceil+1$$

iterations.

Proof. First we upper bound the training error by

$$\begin{aligned}
R_n\left(f^{(T)}\right) &\triangleq \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\left\{\operatorname{sign}\left(f^{(T)}(\mathbf{x}_i)\right)\neq y_i\right\}=\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\left\{f^{(T)}(\mathbf{x}_i)y_i<0\right\}\\
&\leq \frac{1}{n}\sum_{i=1}^{n}\exp\left(-f^{(T)}(\mathbf{x}_i)y_i\right) &(2.15)\\
&= \prod_{t=1}^{T}2\sqrt{\varepsilon^{(t)}(1-\varepsilon^{(t)})}=\prod_{t=1}^{T}\sqrt{1-\left(1-2\varepsilon^{(t)}\right)^2} &(2.16)\\
&\leq \left(\sqrt{1-4\delta^2}\right)^T &(2.17)\\
&\leq \exp\left(-2T\delta^2\right). &(2.18)
\end{aligned}$$

In (2.15) we use $\mathbb{I}\{x<0\}\leq e^{-x}$, (2.16) was shown in (2.13), in (2.17) we apply the assumption that $\varepsilon^{(t)}<\frac{1}{2}-\delta$, and (2.18) follows from $1-x\leq e^{-x}$. The theorem then follows from the discrete nature of the training error: once the upper bound $\exp\left(-2T\delta^2\right)$ falls below $\frac{1}{n}$, the error $R_n\left(f^{(T)}\right)$ must be 0. □

The main significance of Theorem 1 is that it gives a positive answer to PAC learning's (*** explain in the intro) boosting conjecture: it is possible to construct a strong learner (with risk close to 0) by combining a small number of weak learners (with risks only slightly smaller than $\frac{1}{2}$). The exponential convergence speed of the training error, implying that the number of weak learners is $\sim\ln n$, plays a crucial role in proving the conjecture. We will give the formal statement and further discuss its significance in Chapter **??**.

The upper bound (2.16) actually explains the base learner's objective of minimizing $\varepsilon^{(t)}$. It is also clear that by minimizing $\varepsilon^{(t)}$, we do not directly minimize the training error $R_n\left(f^{(T)}\right)$, only through its upper bound

$$R_n^{(e)}\left(f^{(T)}\right)\triangleq\frac{1}{n}\sum_{i=1}^{n}\exp\left(-f^{(T)}(\mathbf{x}_i)y_i\right) \tag{2.19}$$

called the *exponential empirical risk*. Minimizing such convex upper bounds instead of the training error is a common approach in machine learning for two reasons. First, minimizing $R_n\left(f^{(T)}\right)$ is often computationally hard even over simple function classes. Second, minimizing monotonically decreasing margin losses is often justified by probabilistic arguments: we achieve larger margins and better generalization error (risk) by "smoothing" the empirical risk.

The upper bound (2.16) also illustrates another important feature of ADABOOST: the greedy nature of its underlying optimization. Indeed, we do not *globally* minimize the exponential risk $R_n^{(e)}\left(f^{(T)}\right)$ over the linear combinations of $\mathcal{H}$, rather, in each iteration t we add the base learner $h^{(t)}$ and its coefficient $\alpha^{(t)}$ which is *optimal given the classifier* $f^{(t-1)}$ constructed so far. This general interpretation of ADABOOST of greedily optimizing the exponential risk yielded several generalizations and extensions. We will prove the greedy optimization statement formally in Section **??**, and show that under certain conditions this local optimization does lead to the global minimum of $R_n^{(e)}\left(f^{(T)}\right)$.

The condition (2.14) is actually stronger than it seems to be: even for a given data set it requires that the base error is bounded away from $\frac{1}{2}$ for *any* weight distribution $\mathbf{w}^{(t)}$. In fact it can happen easily that (2.14) cannot be satisfied: all we need is a data set $\mathcal{D}_n$ that cannot be separated by any linear combination over $\mathcal{H}$. In this case the training error can obviously not go to 0, and the base errors $\varepsilon^{(t)}$ actually tend to $\frac{1}{2}$. In the limit, the weight distribution is such that for *all* $h \in \mathcal{H}$, the base error is $\frac{1}{2}$. On the other hand, if $\mathcal{D}_n$ is separable by a linear combination over $\mathcal{H}$, then we can prove that (2.14) can always be satisfied, and the actual value of δ (and so the convergence speed) can be related to the separating *margin* between the two classes. We will formalize this statement in Section **??**.

The term $1-2\varepsilon^{(t)}$ that appears in (2.16) is called the *edge* of the base classifier $h^{(t)}$, denoted by $\gamma^{(t)}$. In a certain sense it is a more natural quantity than the error $\varepsilon^{(t)}$, and it will ease the notation in several statements and derivations. It is easy to see that

$$\begin{aligned}
\gamma^{(t)} &\triangleq 1-2\varepsilon^{(t)} = 1-2\sum_{i=1}^{n} w_i^{(t)} \mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\} \\
&= \sum_{i=1}^{n} w_i^{(t)} \left(1-2\mathbb{I}\left\{h^{(t)}(\mathbf{x}_i) \neq y_i\right\}\right) && (2.20)\\
&= \sum_{i=1}^{n} w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i, && (2.21)
\end{aligned}$$

where (2.20) follows from $\sum_{i=1}^{n} w_i^{(t)} = 1$ and (2.21) is true because both y_i and $h^{(t)}(\mathbf{x}_i)$ are $\{-1,1\}$-valued. The edge of a random decision is 0 (on average), so $\gamma^{(t)}$ quantifies how much $h^{(t)}$ is better than a random decision (its "edge" over a random decision). Several formulae of the basic algorithm can either be simplified or they become more "symmetric" using the edge instead of the error. For example, the base coefficient (2.3) can be rewritten as

$$\alpha^{(t)} = \frac{1}{2}\ln\left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}}\right), \tag{2.22}$$

the weight update in lines 7-10 simplifies to

$$w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{1+\gamma^{(t)} h^{(t)}(\mathbf{x}_i) y_i},$$

and the exponential risk (2.13) can be expressed as

$$R_n^{(e)}\left(f^{(T)}\right) = \frac{1}{n}\sum_{i=1}^{n} \exp\left(-f^{(t)}(\mathbf{x}_i) y_i\right) = \prod_{t=1}^{T} \sqrt{\left(1+\gamma^{(t)}\right)\left(1-\gamma^{(t)}\right)} = \prod_{t=1}^{T} \sqrt{1-\gamma^{(t)2}} \tag{2.23}$$

The definition of the margin (2.11) provides another interpretation of (2.21): the edge is the (weighted) *average margin* of the base classifier $h^{(t)}$. In fact the edge is intimately related to margin, and we will elaborate the game-theoretical interplay between the two in Section **??**.

As a final remark, note that the upper bound (2.18) can be sharpened by using the a-posteriori edges $\gamma^{(t)}$ instead of their a-priori lower bound 2δ to

$$R_n\left(f^{(T)}\right) \leq \exp\left(-\frac{1}{2}\sum_{t=1}^{T} \gamma^{(t)2}\right), \tag{2.24}$$

so ADABOOST is guaranteed to achieve 0 training error once $\sum_{t=1}^{T} {\gamma^{(t)}}^2$ exceeds $2\ln n$.

2.4 Base classifiers

The algo doesn't have to be perfect: we don't need the minimum. The design of the base classifier set can be determines by a-priori, domain dependent knowledge.

2.4.1 Decision stumps

If there is no particular a-priori knowledge available on the domain of the learning problem, small decision trees or, in the extreme case, *decision stumps* (decision trees with two leaves) are often used. A decision stump can be defined by three parameters, the index j of the feature[4] that it cuts, the threshold θ of the cut, and the sign of the decision. Formally, a *positive stump* is defined by

$$h_{j,\theta+}(\mathbf{x}) \triangleq 2I_{\{x^{(j)} \geq \theta\}} - 1 = \begin{cases} 1 & \text{if } x^{(j)} \geq \theta, \\ -1 & \text{otherwise,} \end{cases} \tag{2.25}$$

and a *negative stump* is defined by

$$h_{j,\theta-}(\mathbf{x}) \triangleq -h_{j,\theta+}(\mathbf{x}) = 2I_{\{x^{(j)} < \theta\}} - 1 = \begin{cases} 1 & \text{if } x^{(j)} < \theta, \\ -1 & \text{otherwise.} \end{cases} \tag{2.26}$$

Although decision stumps may seem very simple, when boosted, they yield good classifiers in practice. The algorithm we describe here can also be used as a basic building block for more complex base learners, such as trees or products.

The optimal decision stump (that minimizes the the base error $\varepsilon^{(t)}$ (2.2)) can be found in $O(nd)$ time using exhaustive search (Figure 2.3). The pseudocode is slightly simpler if we maximize the edge $\gamma^{(t)}$ (2.21) instead of minimizing the error $\varepsilon^{(t)}$, so we adopt this notation. First, all feature vectors $\mathbf{x}^{(j)} = \left(x_1^{(j)}, \ldots, x_n^{(j)}\right)$ must be sorted in increasing order. Technically this should be done by creating d indirect index sets and storing the order permutations. This step precedes the boosting loop, so the overall complexity of boosting decision stumps is $O(dn\log n + Tdn) = O\big(dn(\log n + T)\big)$. To ease the notation, we assume that the feature vectors are sorted, that is, $x_1^{(j)} \leq \ldots \leq x_n^{(j)}$ for all $j = 1.\ldots,d$.

The algorithm will examine all possible cutting thresholds θ between non-identical consecutive feature values $x_{i-1}^{(j)}$ and $x_i^{(j)}$. We cut halfway between the feature values, so for all $i = 2,\ldots,n$ and $j = 1,\ldots,d$,

$$\theta_i^{(j)} = \frac{x_i^{(j)} + x_{i-1}^{(j)}}{2}.$$

Since we have $O(nd)$ thresholds and computing the edge $\gamma^{(t)}$ requires $O(n)$ operations, the naive algorithm would take $O(n^2 d)$ time to find the best stump. In Figure 2.3 we use the fact that the edge of "consecutive" stumps can be updated in constant time, so we can find the best stump in linear time using one sweep over data points and features.

For each feature, we start with the constant classifier $h_0(\mathbf{x}) \equiv 1$ (which is equivalent to a positive stump with $\theta < x_1^{(j)}$) and set the edge γ in line 4 to the edge of the constant classifier

$$\gamma_0 = \sum_{i=1}^{n} w_i y_i.$$

[4] We assume that features are real valued, so observations $\mathbf{x}$ are in $\mathbb{R}^d$.

```
DecisionStump(𝒟_n, w)

1  γ_0 ← Σ_{i=1}^{n} w_i y_i          ▷ edge of constant classifier h_0(x) ≡ 1
2  γ* ← γ_0                           ▷ best edge
3  for j ← 1 to d                     ▷ all (numeric) features
4      γ ← γ_0                        ▷ edge of the constant classifier
5      for i ← 2 to n                 ▷ all points in order x_1^(j) ≤ ... ≤ x_n^(j)
6          γ ← γ − 2 w_{i−1} y_{i−1}  ▷ update edge of positive stump
7          if x_{i−1}^(j) ≠ x_i^(j) then   ▷ no threshold if identical coordinates
8              if |γ| > |γ*| then     ▷ found better stump
9                  γ* ← γ             ▷ update best edge
10                 j* ← j             ▷ update index of best feature
11                 θ* ← (x_i^(j) + x_{i−1}^(j)) / 2   ▷ update best threshold
12 if γ* = γ_0                        ▷ did not beat the constant classifier
13     return sign(γ_0) × h_0         ▷ ± constant classifier
14 else
15     return sign(γ*) × h_{j*,θ*+}   ▷ best stump
```

Figure 2.3: Exhaustive search for the best decision stump. We assume that the coordinates $x_1^{(j)},\ldots,x_n^{(j)}$ are sorted for each feature j. All thresholds θ halfway between non-identical coordinates are examined.

The edge γ is then updated in line 6 by "moving" the next feature value $x_{i-1}^{(j)}$ to the left side of the threshold θ (Figure 2.4). Since we cannot cut between two identical feature values, we only examine the edge if a new stump can be defined (line 7). To use only one sweep, we check for negative and positive stumps at the same time, so we update the edge, the threshold, and the feature index if the absolute value of γ is larger than the absolute value of the currently best γ^* (line 8). At the end of the algorithm we first check if any of the stumps has beaten the constant classifier (line 12), then return either a positive or a negative stump, depending on the sign of γ^*.

*** The same one-sweep approach can be done for other losses such as entropy, Gini-index, quadratic, but, interestingly, not for the absolute error (in case of regression stumps) *exercise*.

2.5 A simple example

In this section we illustrate the theoretical concepts by executing AdaBoost with decision stumps on a simple example. We generate 200 two-dimensional observations $\mathbf{x}_i = \left(x_i^{(1)}, x_i^{(2)}\right)$ uniformly in a unit square, and, to pose a slight challenge to decision stumps, we define the classes by a diagonal cut (Figure 2.5). Formally, $y_i = 1$ if an only if $x_i^{(1)} + x_i^{(2)} \leq 1$.

The execution of AdaBoost can be followed in Figure 2.6. We plot the data points and the decision regions at $t = 1,3,5,10,20,40$ iterations. The areas of the disks are proportional to the weights $w_i^{(t)}$. It can be seen very well that the weights of "hard" data points near the decision boundary will dominate as the AdaBoost progresses. Figure 2.7 shows the same situation after $t = 40$ iterations, but this this time we modulate the color intensities of the decision regions by the margin $\rho_{\mathbf{x},y}(f) = f^{(t)}(x)y$. Naturally, far from the decision boundary the classification is very confident, whereas the light color indicates small margins close to the diagonal decision boundary. The plot

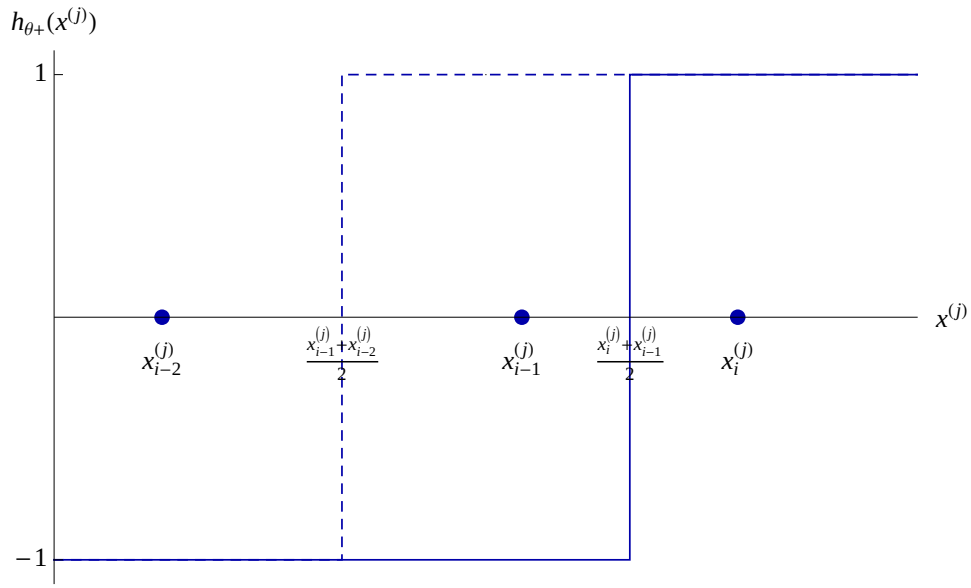


Figure 2.4: Updating the edge γ in line 6 of DECISIONSTUMP$(\mathcal{D}_n, \mathbf{w})$. If $y_{i-1}^{(j)} = 1$, then γ decreases by $2w_{i-1}$, and if $y_{i-1}^{(j)} = -1$, then γ increases by $2w_{i-1}$.

also illustrates the formal relationship (2.12) between the weights $w_i^{(t)}$ and the margin $\rho_{\mathbf{x},y}(f)$: first, the smaller the margin, the larger are the weights, and second, the weights only depend on the margin, so the disks have an identical size in constant intensity rectangles.

In Figure 2.8 we plot the training error and its two bounds as a function of the number of iterations. Figure 2.8(a) indicates that the training error becomes 0 after 26 iterations. In Section 2.3 we showed that ADABOOST attempts to minimize the exponential risk $R_n^{(e)}\left(f^{(t)}\right)$ (2.23) in a greedy fashion; the red curve in Figure 2.8(b) indicates that $R_n^{(e)}\left(f^{(t)}\right)$ is a rather tight upper bound of the training error. On the other hand, the a-priori bound[5] (2.17), used to establish the convergence speed independently of the algorithm's dynamics, may be quite loose, as indicated by the green curve in Figure 2.8(b).

2.6 Discussion

For the neurally inclined: it's like constructing a one-hidden-layer neural net incrementally.

The entropy – info-theoretical view.

The game-theoretical view.

Boosting as feature selector.

2.7 Bibliographical notes

[5]In the case of a finite base classifier class $\mathcal{H}$, the constant δ that guarantees a universal upper bound $\frac{1}{2} - \delta$ on the base error $\varepsilon^{(t)}$ can be determined using linear programming; see Section **??**.

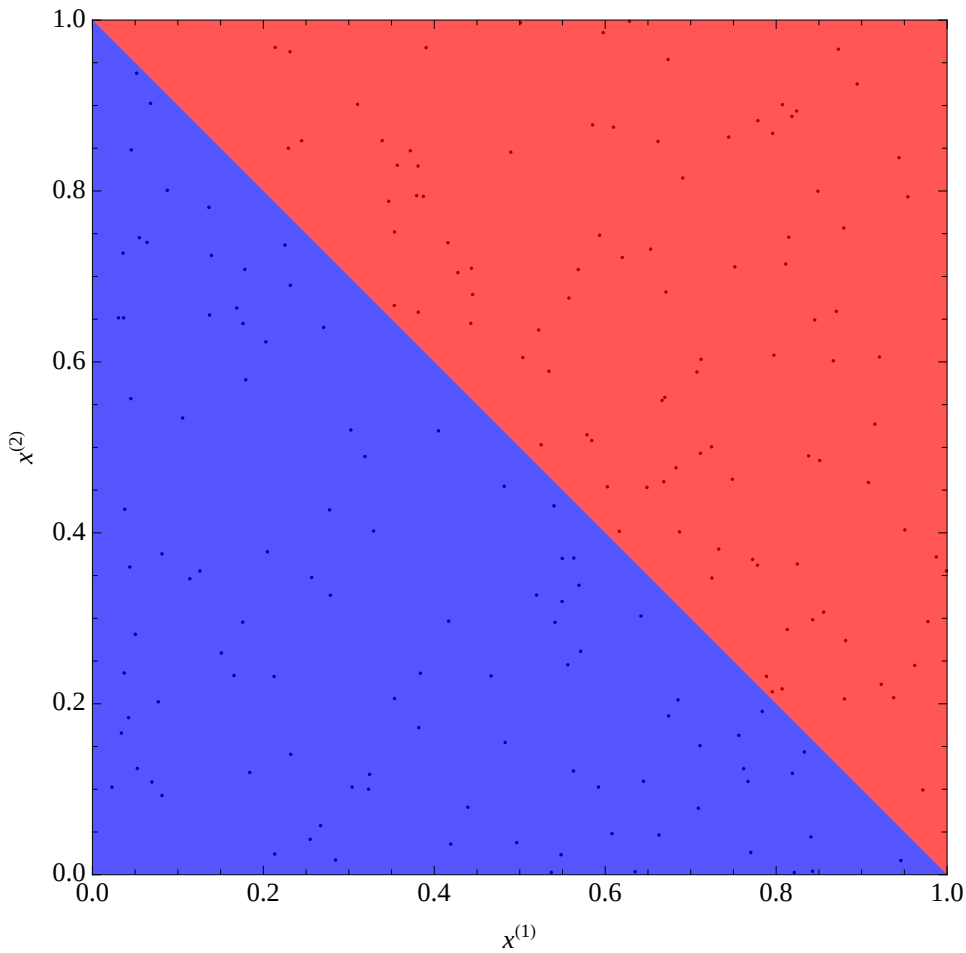
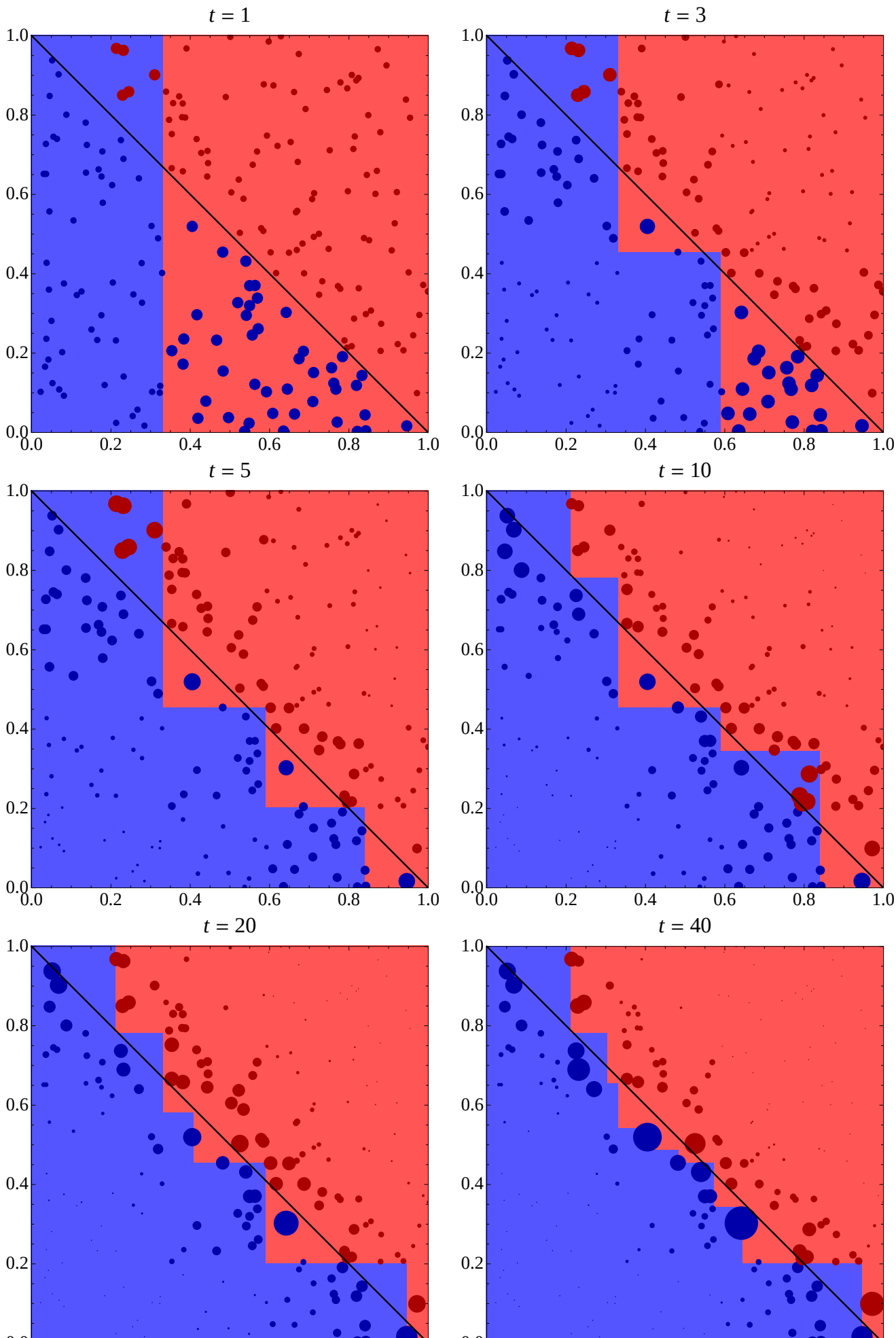


Figure 2.5: Observations $\mathbf{x}_i = \left(x_i^{(1)}, x_i^{(2)}\right)$ are generated uniformly in a unit square; class labels are defined by $y_i = 1$ iff $x_i^{(1)} + x_i^{(2)} \leq 1$. Positive points (and region) are blue, and negative points are red.


t = 1
t = 3
t = 5
t = 10
t = 20
t = 40
0.0
0.2
0.4
0.6
0.8
1.0

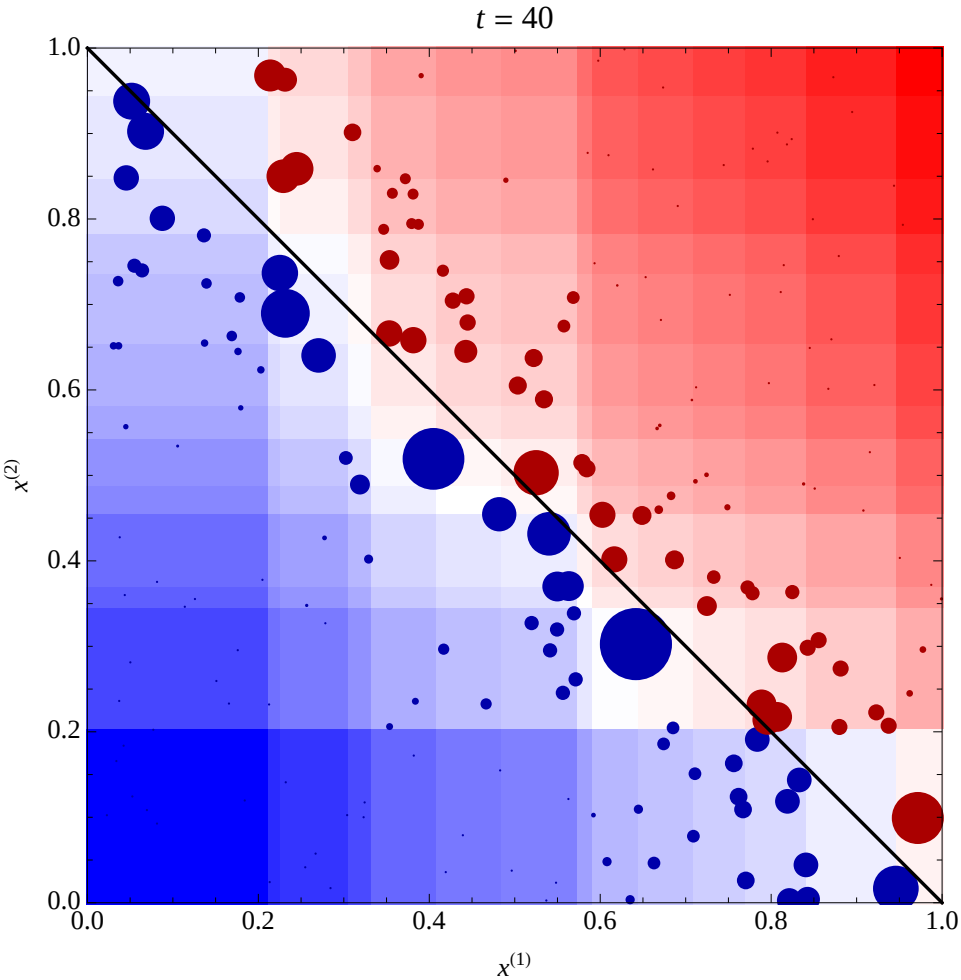
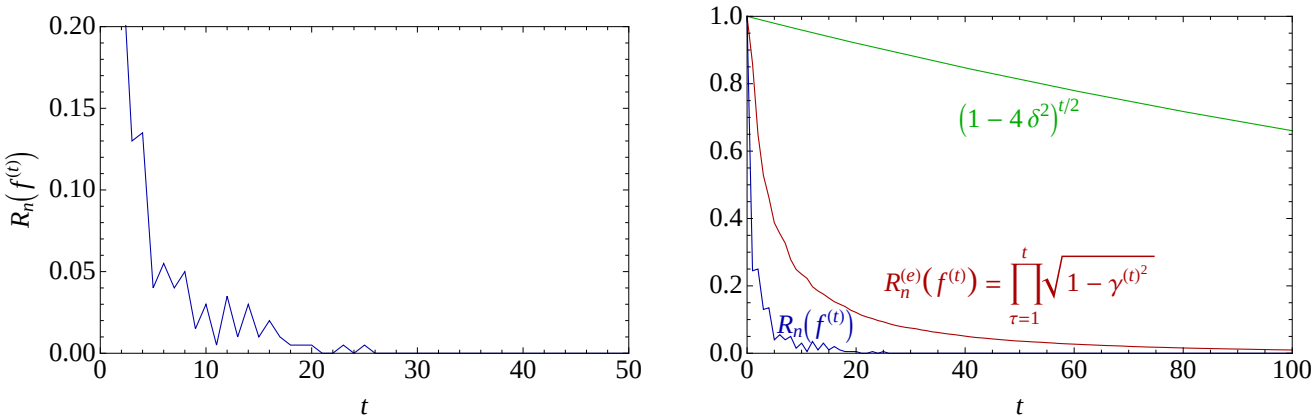


Figure 2.7: The weights and the margins after $t = 40$ iterations. The color intensity is proportional to the margin $\rho_{\mathbf{x},y}(f) = f^{(t)}(x)y$.



(a) (b)

Figure 2.8: (a) The training error $R_n\left(f^{(t)}\right)$ and (b) its bounds. The green curve is the a-priori bound (2.17) and the red curve is the exponential risk $R_n^{(e)}\left(f^{(t)}\right)$ (2.23).

Bibliography

[FS97] 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*, 55:119–139, 1997.