Finally, since the VC-dimension of decision stumps is 2, if the sample size is greater than $\Omega(\log(1/\delta)/\epsilon^2)$, then with probability of at least $1-\delta$, the ERM_B rule returns a hypothesis with an error of at most $1/3+\epsilon$. Setting $\epsilon = 1/12$ we obtain that the error of ERM_B is at most $1/3+1/12 = 1/2-1/12$.

We see that ERM_B is a γ-weak learner for $\mathcal{H}$. We next show how to implement the ERM rule efficiently for decision stumps.

10.1.1 Efficient Implementation of ERM for Decision Stumps

Let $\mathcal{X} = \mathbb{R}^d$ and consider the base hypothesis class of decision stumps over $\mathbb{R}^d$, namely,

$$\mathcal{H}_{\text{DS}} = \{\mathbf{x} \mapsto \text{sign}(\theta - x_i)\cdot b : \theta \in \mathbb{R}, i \in [d], b \in \{\pm 1\}\}.$$

For simplicity, assume that $b = 1$; that is, we focus on all the hypotheses in $\mathcal{H}_{\text{DS}}$ of the form $\text{sign}(\theta - x_i)$. Let $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ be a training set. We will show how to implement an ERM rule, namely, how to find a decision stump that minimizes $L_S(h)$. Furthermore, since in the next section we will show that AdaBoost requires finding a hypothesis with a small risk relative to some distribution over S, we will show here how to minimize such risk functions. Concretely, let $\mathbf{D}$ be a probability vector in $\mathbb{R}^m$ (that is, all elements of $\mathbf{D}$ are nonnegative and $\sum_i D_i = 1$). The weak learner we describe later receives $\mathbf{D}$ and S and outputs a decision stump $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the risk w.r.t. $\mathbf{D}$,

$$L_{\mathbf{D}}(h) = \sum_{i=1}^{m} D_i \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]}.$$

Note that if $\mathbf{D} = (1/m, \ldots, 1/m)$ then $L_{\mathbf{D}}(h) = L_S(h)$.

Recall that each decision stump is parameterized by an index $j \in [d]$ and a threshold θ. Therefore, minimizing $L_{\mathbf{D}}(h)$ amounts to solving the problem

$$\min_{j\in[d]} \min_{\theta\in\mathbb{R}} \left(\sum_{i:y_i=1} D_i \mathbb{1}_{[x_{i,j}>\theta]} + \sum_{i:y_i=-1} D_i \mathbb{1}_{[x_{i,j}\leq\theta]} \right). \tag{10.1}$$

Fix $j \in [d]$ and let us sort the examples so that $x_{1,j} \leq x_{2,j} \leq \ldots \leq x_{m,j}$. Define $\Theta_j = \{\frac{x_{i,j}+x_{i+1,j}}{2} : i \in [m-1]\} \cup \{(x_{1,j}-1), (x_{m,j}+1)\}$. Note that for any $\theta \in \mathbb{R}$ there exists $\theta' \in \Theta_j$ that yields the same predictions for the sample S as the threshold θ. Therefore, instead of minimizing over $\theta \in \mathbb{R}$ we can minimize over $\theta \in \Theta_j$.

This already gives us an efficient procedure: Choose $j \in [d]$ and $\theta \in \Theta_j$ that minimize the objective value of Equation (10.1). For every j and $\theta \in \Theta_j$ we have to calculate a sum over m examples; therefore the runtime of this approach would be $O(dm^2)$. We next show a simple trick that enables us to minimize the objective in time $O(dm)$.

The observation is as follows. Suppose we have calculated the objective for $\theta \in (x_{i-1,j}, x_{i,j})$. Let $F(\theta)$ be the value of the objective. Then, when we consider $\theta' \in (x_{i,j}, x_{i+1,j})$ we have that

$$F(\theta') = F(\theta) - D_i \mathbb{1}_{[y_i=1]} + D_i \mathbb{1}_{[y_i=-1]} = F(\theta) - y_i D_i.$$

Therefore, we can calculate the objective at θ' in a constant time, given the objective at the previous threshold, θ. It follows that after a preprocessing step in which we sort the examples with respect to each coordinate, the minimization problem can be performed in time $O(dm)$. This yields the following pseudocode.

ERM for Decision Stumps

input:
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
distribution vector $\mathbf{D}$
goal: Find $j^\star, \theta^\star$ that solve Equation (10.1)
initialize: $F^\star = \infty$
for $j = 1, \ldots, d$
sort S using the j'th coordinate, and denote
$x_{1,j} \leq x_{2,j} \leq \cdots \leq x_{m,j} \leq x_{m+1,j} \stackrel{\text{def}}{=} x_{m,j} + 1$
$F = \sum_{i:y_i=1} D_i$
if $F < F^\star$
$F^\star = F, \theta^\star = x_{1,j} - 1, j^\star = j$
for $i = 1, \ldots, m$
$F = F - y_i D_i$
if $F < F^\star$ and $x_{i,j} \neq x_{i+1,j}$
$F^\star = F, \theta^\star = \frac{1}{2}(x_{i,j} + x_{i+1,j}), j^\star = j$
output $j^\star, \theta^\star$

10.2 ADABOOST

AdaBoost (short for Adaptive Boosting) is an algorithm that has access to a weak learner and finds a hypothesis with a low empirical risk. The AdaBoost algorithm receives as input a training set of examples $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where for each i, $y_i = f(\mathbf{x}_i)$ for some labeling function f. The boosting process proceeds in a sequence of consecutive rounds. At round t, the booster first defines a distribution over the examples in S, denoted $\mathbf{D}^{(t)}$. That is, $\mathbf{D}^{(t)} \in \mathbb{R}^m_+$ and $\sum_{i=1}^m D_i^{(t)} = 1$. Then, the booster passes the distribution $\mathbf{D}^{(t)}$ and the sample S to the weak learner. (That way, the weak learner can construct i.i.d. examples according to $\mathbf{D}^{(t)}$ and f.) The weak learner is assumed to return a "weak" hypothesis, h_t, whose error,

$$\epsilon_t \stackrel{\text{def}}{=} L_{\mathbf{D}^{(t)}}(h_t) \stackrel{\text{def}}{=} \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]},$$

is at most $\frac{1}{2} - \gamma$ (of course, there is a probability of at most δ that the weak learner fails). Then, AdaBoost assigns a weight for h_t as follows: $w_t = \frac{1}{2} \log\left(\frac{1}{\epsilon_t} - 1\right)$. That is, the weight of h_t is inversely proportional to the error of h_t. At the end of the round, AdaBoost updates the distribution so that examples on which h_t errs will get a higher probability mass while examples on which h_t is correct will get a lower probability mass. Intuitively, this will force the weak learner to focus on the problematic examples in the next round. The output of the AdaBoost algorithm is a