Recitation 9: December 28

Lecturer: Regev Schweiger *Scribe: Yishay Mansour*

9.1 AdaBoost

9.1.1 Review

Reminder: Algorithm A learns Weak-PAC a concept class C with H if:
$\exists \gamma > 0$,
$\forall c^* \in C$, (target function)
$\forall D$, (distribution)
$\forall \delta > \frac{1}{2}$,
With probability $1-\delta$, algorithm A outputs an hypothesis $h \in H$ such that $error(h) \leq \frac{1}{2} - \gamma$.

Input: A set of m classified examples: $S = \{\langle \mathbf{x}_1, y_1 \rangle, \langle \mathbf{x}_2, y_2 \rangle, \cdots, \langle \mathbf{x}_m, y_m \rangle\}$ where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$ and a class of weak learners H.
Definitions: Let D_t denote the distribution (weights) of the examples at iteration t: $D_t(x_i)$ is the weight of example $\langle \mathbf{x}_i, y_i \rangle$ at iteration t.
Initialization:

$$D_1(\mathbf{x}_i) = \frac{1}{m} \quad \forall i \in \{1, \cdots, m\}$$

Iterate: $t = 1, 2, \ldots, T$

$$\begin{aligned} h_t &= \arg\min_{h \in H} \Pr_{\mathbf{x} \sim D_t} [h(\mathbf{x}) \neq y] \\ \epsilon_t &= \Pr_{\mathbf{x} \sim D_t} [h_t(\mathbf{x}) \neq y] \\ \alpha_t &= \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t} \\ D_{t+1}(\mathbf{x}_i) &= \frac{D_t(\mathbf{x}_i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \end{aligned}$$

Finally:

$$H(\mathbf{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

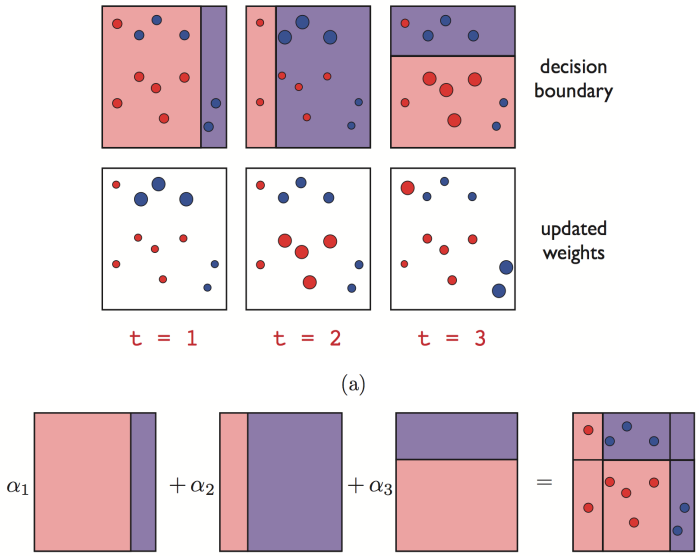

Figure 9.1: Example of AdaBoost with axis-aligned hyperplanes as base learners. (a) The top row shows decision boundaries at each boosting round. The bottom row shows how weights are updated at each round, with incorrectly (resp., correctly) points given increased (resp., decreased) weights. The size of the points represents the distribution weight assigned to them at each boosting round. (b) Visualization of final classifier, constructed as a linear combination of base learners.[1]

Figure 9.1 illustrates the AdaBoost algorithm.

9.1.2 Feature decision stumps as weak learners

We show how to efficiently compute the optimal weak learner in the first step of each AdaBoost iteration for a class H of feature decision stumps, given the distribution/weights $D(x_i)$ over the sample set:

Assume each example $\mathbf{x}$ has a set of features $f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}), \ldots$. Such features can be one of a few types:

1. Binary features: $f_k(\mathbf{x}) \in \{0, 1\}$.
2. Discrete features: for example $f_k(\mathbf{x}) \in \{a, b, c\}$.
3. Continuous features: $f_k(\mathbf{x}) \in \mathbb{R}$.

[1] Adapted from *Foundations of Machine Learning*

Binary Features

We have *decision stumps* which have two parameters $c_0, c_1 \in \{+1, -1\}$. Let us fix a binary feature $f_k(\mathbf{x})$ [2]. Then a stump has the form

$$h^k_{c_0,c_1}(\mathbf{x}) = \begin{cases} c_0 & f_k(\mathbf{x}) = 0 \\ c_1 & f_k(\mathbf{x}) = 1 \end{cases}$$

Given a distribution $D(\mathbf{x}_i)$ we select the optimal c_0 and c_1 for h^k (which is based on feature f_k) as follows. For $j \in \{0, 1\}$ and $b \in \{+1, -1\}$ we compute

$$w^j_b = \sum_{\{i|f_k(x_i)=j, y_i=b\}} D(\mathbf{x}_i)$$

Note that those values may be computed during a single pass over the m samples. We have that the error of $h^k_{c_0,c_1}$ is

$$error(h^k_{c_0,c_1}) = w^0_{-c_0} + w^1_{-c_1}$$

This is because $h^k_{c_0,c_1}$ errs on samples $\langle \mathbf{x}_i, -c_0 \rangle$ with $f_k(\mathbf{x}_i) = 0$ or samples $\langle \mathbf{x}_i, -c_1 \rangle$ with $f_k(\mathbf{x}_i) = 1$. Therefore the optimal stump $h^k_{c_0,c_1}$ has

$$c_j = \begin{cases} +1 & w^j_{-1} \leq w^j_{+1} \\ -1 & w^j_{-1} > w^j_{+1} \end{cases}$$

Discrete Features

The feature $f_k(\mathbf{x})$ has ℓ different values, i.e., $f_k(\mathbf{x}) \in \{v_1, , \ldots, v_\ell\}$. Similarly, we can set the weak learner to be

$$h^k_{c_1,\ldots,c_\ell}(\mathbf{x}) = \begin{cases} c_1 & f_k(\mathbf{x}) = v_1 \\ \vdots & \vdots \\ c_\ell & f_k(\mathbf{x}) = v_\ell \end{cases}$$

Again, the parameters $c_j \in \{+1, -1\}$. As before we can set (during a single pass over the sample set), for $j \in [1, \ell]$ and $b \in \{+1, -1\}$ we compute

$$w^j_b = \sum_{\{i:|f_k(\mathbf{x}_i)=v_j, y_i=b\}} D(\mathbf{x}_i)$$

The error is

$$error(h^k_{c_1,\ldots,c_\ell}) = \sum_{j=1}^{\ell} w^j_{-c_j}$$

and we select c_j as before.

[2] Once we find the optimal stump for each k, an outer loop will choose the best across the possible values of k. Therefore, in what follows we assume a fixed k.

Continuous Features

We now have that $f_k(\mathbf{x}) \in \mathbb{R}$. A possible form for a weak learner is

$$h_v^k(\mathbf{x}) = \begin{cases} c_0 & f_k(\mathbf{x}) \leq v \\ c_1 & f_k(\mathbf{x}) > v \end{cases}$$

Technically v can be of an infinite number of values. However, if we have a sample of size m we can sort the values $f_k(\mathbf{x}_1) \leq \cdots \leq f_k(\mathbf{x}_m)$. Although there are an infinite number of possible values for v, there are really only $m+1$ interesting ones. Namely, $v_0, \ldots, v_m$, where $v_0 \leq f_k(\mathbf{x}_1)$, $v_j \in (f_k(\mathbf{x}_j), f_k(\mathbf{x}_{j+1}]$ and $v_m > f_k(\mathbf{x}_m)$ [3].

Now, for any choice of $m+1$ such values for v we can apply the method used for the case of binary features above (replacing the binary condition over values of b, $f_k(\mathbf{x}) = 0$, $f_k(\mathbf{x}) = 1$ with the two options $f_k(\mathbf{x}) \leq v$, $f_k(\mathbf{x}) > v$) to find the optimal c_0, c_1.

Observe, however, that while boosting stumps are widely used in combination with AdaBoost and can perform well in practice, the algorithm that returns the stump with the minimal empirical error is not a weak learner! Consider, for example, the simple XOR example with four data points lying in R2, where points in the second and fourth quadrants are labeled positively and those in the first and third quadrants negatively. Then, no decision stump can achieve an accuracy better than 1/2.

9.1.3 Error of h_t on D_{t+1}

We will show that the hypothesis h_t has error 1/2 on D_{t+1} and hence $h_{t+1} \neq h_t$ (since by the weak learning assumption, the weak learner has error bounded away from $\frac{1}{2}$).

Claim 9.1 $\Pr_{\mathbf{x}_i \sim D_{t+1}}[h_t(\mathbf{x}_i) \neq y_i] = \frac{1}{2}$

We first show a few quantities that we will use later.

$$e^{\alpha_t} = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \qquad e^{-\alpha_t} = \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}$$

Therefore,

$$\epsilon_t e^{\alpha_t} = \sqrt{\epsilon_t(1-\epsilon_t)} = (1-\epsilon_t)e^{-\alpha_t}$$

Recall that,

$$\sum_{\{i|y_i \neq h_t(\mathbf{x}_i)\}} D_t(\mathbf{x}_i) = \epsilon_t.$$

[3] For example, any two values v and u in the range $[f_k(\mathbf{x}_j), f_k(\mathbf{x}_{j+1}]$ are equivalent in terms of the values $h_v(\cdot)$ and $h_u(\cdot)$ over the m samples.

Now the normalization is,

$$\begin{aligned} Z_t &= \sum_{i=1}^{m} D_t(\mathbf{x}_i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)} \\ &= \sum_{\{i|y_i=h_t(\mathbf{x}_i)\}} D_t(\mathbf{x}_i) e^{-\alpha_t} + \sum_{\{i|y_i\neq h_t(\mathbf{x}_i)\}} D_t(\mathbf{x}_i) e^{\alpha_t} \\ &= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \\ &= 2\sqrt{\epsilon_t(1-\epsilon_t)} \end{aligned}$$

Now we can prove the claim.

$$\begin{aligned} \Pr_{\mathbf{x}\sim D_{t+1}}[h_t(\mathbf{x}) \neq y] &= \sum_{\{i|y_i\neq h_t(\mathbf{x}_i)\}} D_{t+1}(\mathbf{x}_i) \\ &= \sum_{\{i|y_i\neq h_t(\mathbf{x}_i)\}} \frac{D_t(\mathbf{x}_i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \\ &= \sum_{\{i|y_i\neq h_t(\mathbf{x}_i)\}} D_t(\mathbf{x}_i) \frac{e^{\alpha_t}}{Z_t} \\ &= \frac{e^{\alpha_t}}{Z_t}\epsilon_t = \frac{\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}\epsilon_t}{2\sqrt{\epsilon_t(1-\epsilon_t)}} = \frac{1}{2} \end{aligned}$$

9.1.4 Coordinate Descent and AdaBoost

AdaBoost was designed to address a novel theoretical question, that of designing a strong learning algorithm using a weak learning algorithm. We will show, however, that it coincides in fact with a very simple and classical algorithm, which consists of applying a coordinate descent technique to a convex and differentiable objective function. Coordinate descent is just like gradient descent, except that you can't move along the gradient, you have to choose just one coordinate at a time to move along.

Suppose (for the sake of simplicity) that there is a finite number of weak classifiers, $h_j(\mathbf{x})$, $j = 1, \ldots, n$. We wish to find a linear combination with coefficients λ_j, that minimizes the average exponential loss:

$$F(\boldsymbol{\lambda}) = F(\lambda_1, \ldots, \lambda_n) = \frac{1}{m}\sum_{i=1}^{m} e^{-y_i \sum_{j=1}^{n} \lambda_j h_j(\mathbf{x}_i)}$$

The coordinate descent algorithm will first choose a coordinate k in which the slope is

the steepest. This translates to calculating, for each k, the following:

$$\begin{aligned}\frac{\partial}{\partial\alpha}F(\boldsymbol{\lambda}+\alpha\cdot\mathbf{e}_k) &= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\alpha}\left(e^{-y_i\sum_{j=1}^{n}(\lambda_j+\alpha\cdot\delta_{j=k})h_j(\mathbf{x}_i)}\right)\\ &= -\frac{1}{m}\sum_{i=1}^{m}y_ih_k(\mathbf{x}_i)e^{-y_i\sum_{j=1}^{n}(\lambda_j+\alpha\cdot\delta_{j=k})h_j(\mathbf{x}_i)}\end{aligned}$$

Substituting $\alpha=0$ and defining $D(\mathbf{x}_i)=e^{-y_i\sum_{j=1}^{n}\lambda_jh_j(\mathbf{x}_i)}/Z$, with Z the relevant normalizing factor, gives:

$$\begin{aligned}&= -\frac{1}{m}\sum_{i=1}^{m}y_ih_k(\mathbf{x}_i)e^{-y_i\sum_{j=1}^{n}\lambda_jh_j(\mathbf{x}_i)}\\ &= -\frac{1}{m}\sum_{i=1}^{m}y_ih_k(\mathbf{x}_i)D(\mathbf{x}_i)Z\\ &\propto \sum_{i=1}^{m}y_ih_k(\mathbf{x}_i)D(\mathbf{x}_i)\\ &= (1-\epsilon_k)-(\epsilon_k)=1-2\epsilon_k\end{aligned}$$

So finding the direction k the maximizes the slope is equivalent to minimizing ϵ_k, which is what AdaBoost does at each step!

The next stage in this coordinate descent variant is to analytically calculate the optimal step α is direction k. It can be shown that this gives the same α in AdaBoost.