CS260: Machine Learning Theory
Lecture 13: Weak vs. Strong Learning and the Adaboost Algorithm

November 7, 2011

Lecturer: Jennifer Wortman Vaughan

1 Weak vs. Strong Learning

The big idea behind boosting is to construct an accurate hypothesis by combining the predictions of many individual hypotheses, each of which performs slightly better than random guessing on some particular distribution of data. This idea originally arose from attempts to prove robustness of the PAC model, in other words, attempts to prove that small changes to the model definitions don't lead to dramatically different results, such as the ability to learn different classes of functions.

In particular, it grew out of the answer to a question posed in the late 80s by Kearns and Valiant about whether so-called "weak" learning implies "strong" learning (which is simply the usual PAC definition of learning). We therefore begin our discussion of boosting with a review of the PAC model, and a discussion of the differences between these notions of learnability. For simplicity, we'll revert to our most basic definition of PAC learning, which does not yet incorporate things like the size of the input, but all of the results we will discuss hold for our revised definition too.

Definition 1 (Strong PAC Learnability)**.** *A concept class $\mathcal{C}$ is **strongly PAC learnable** using a hypothesis class $\mathcal{H}$ if there exists an algorithm $\mathcal{A}$ such that for any $c \in \mathcal{C}$, for any distribution $\mathcal{D}$ over the input space, for any $\epsilon \in (0, 1/2)$ and $\delta \in (0, 1/2)$, given access to a polynomial (in $1/\epsilon$ and $1/\delta$) number of examples drawn i.i.d. from $\mathcal{D}$ and labeled by c, $\mathcal{A}$ outputs a function $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\mathrm{err}(h) \leq \epsilon$.*

This definition is "strong" in the sense that it requires that $\mathrm{err}(h)$ can be driven arbitrarily close to 0 by choosing an appropriately small value of ϵ. In contrast, weak learnability only requires $\mathcal{A}$ to return a hypothesis that does better than random guessing.

Definition 2 (Weak PAC Learnability)**.** *A concept class $\mathcal{C}$ is **weakly PAC learnable** using a hypothesis class $\mathcal{H}$ if there exists an algorithm $\mathcal{A}$ and a value $\gamma > 0$ such that for any $c \in \mathcal{C}$, for any distribution $\mathcal{D}$ over the input space, for any $\delta \in (0, 1/2)$, given access to a polynomial (in $1/\delta$) number of examples drawn i.i.d. from $\mathcal{D}$ and labeled by c, $\mathcal{A}$ outputs a function $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\mathrm{err}(h) \leq 1/2 - \gamma$.*

We will sometimes refer to γ as the advantage of $\mathcal{A}$ (over random guessing).

It's clear that strong learnability implies weak learnability. The question we want to answer is whether weak learnability implies strong learnability. More formally, we might ask the following:

If $\mathcal{C}$ is weakly learnable using $\mathcal{H}$, must there exist some $\mathcal{H}'$ such that $\mathcal{C}$ is (strongly) learnable using $\mathcal{H}'$?

All CS260 lecture notes build on the scribes' notes written by UCLA students in the Fall 2010 offering of this course. Although they have been carefully reviewed, it is entirely possible that some of them contain errors. If you spot an error, please email Jenn.

We can think about this question as follows. Fix an arbitrary $\epsilon > 0$. Suppose we are given a polynomial number (in $1/\delta$ and $1/\epsilon$) of samples drawn i.i.d. from some distribution $\mathcal{D}$ and labeled by a target $c \in \mathcal{C}$, as well as a weak learning algorithm $\mathcal{A}$ for $\mathcal{C}$. Can we incorporate $\mathcal{A}$ into a new algorithm that is guaranteed to produce a new function h such that with high probability, $err(h) \leq \epsilon$?

The first boosting algorithm was developed by Rob Schapire in order to answer this question. This paved the way for the immensely popular AdaBoost algorithm, which was developed by Freund and Schapire a couple of years later.

2 AdaBoost

The AdaBoost (for *adaptive* boosting) algorithm uses a weak learning algorithm $\mathcal{A}$ as a black box. For now, we won't care much about where this weak learning algorithm comes from, and will assume that it is given. In practice, the weak learning algorithms that are used with Adaboost are extremely simple. For example, a weak learner might output the best decision rule based on a single feature (e.g., "predict 1 if and only if the value of the ith feature is negative").

Adaboost takes as input a set of points $(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)$, with $y_i \in \{-1, 1\}$ for all i. It runs for T rounds, calling the weak learning algorithm $\mathcal{A}$ on each round. In order to obtain new information from $\mathcal{A}$ on each round, it is necessary to continue creating new sets of input points to feed to $\mathcal{A}$. Since the original set of m input points is fixed, we do this by fixing new distributions $\mathcal{D}_t$ over these points at each round t, and selecting a polynomial size sample from the each new distribution to give as input to $\mathcal{A}$. Each round, $\mathcal{A}$ returns a function that performs a bit better than random guessing with respect to the particular distribution $\mathcal{D}_t$.

The algorithm is stated formally below. We use $\mathcal{D}_t(i)$ to denote the weight that the distribution at time t places on the ith input point.

The AdaBoost Algorithm:

- Initialize $\mathcal{D}_1(i) = 1/m$ for all $i \in \{1, \cdots, m\}$
- For each round $t = 1, ..., T$
  - Draw a "sufficiently large" (polynomial-size) sample i.i.d. from $\mathcal{D}_t$
  - Run $\mathcal{A}$ on this sample to produce h_t
  - Set $\epsilon_t = \sum_{i=1}^m \mathcal{D}_t(i)\mathbb{I}(h_t(\vec{x}_i) \neq y_i)$
  - Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - Update the distribution, setting
    $$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)e^{-\alpha_t y_i h_t(\vec{x}_i)}}{Z_t}$$
    for all $i \in \{1, \cdots, m\}$, where Z_t is a normalizing factor
- Output a function h with
  $$h(\vec{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\vec{x})\right)$$
  for all $\vec{x}$.

Here the "sufficiently large" sample should be large enough to guarantee that $\mathcal{A}$ outputs a function satisfying the weak learning guarantee with high probability.

Note that the weight updates depend on $y_i h_t(\vec{x}_i)$. If the function output by the weak learner at round t correctly predicts the label of $\vec{x}_i$, then $y_i h_t(\vec{x}_i) = 1$; otherwise $y_i h_t(\vec{x}_i) = -1$. These distribution updates look a lot like the weight updates made by Winnow or Randomized Weighted Majority!

Also note that the normalizing factor Z_t can be calculated as

$$Z_t = \sum_{i=1}^{m} \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\vec{x}_i)} .$$

AdaBoost is popular for a number of reasons:

- It has no tricky parameters to set.
- It does not require prior knowledge of the advantage γ (as in Definition 2) for the weak learner $\mathcal{A}$.
- It is computationally tractable and not too hard to implement.
- It generalizes very well and avoids overfitting.
- It satisfies some nice theoretical guarantees, as we will see later in this lecture and next time.

3 Bounding the Training Error

We will now derive a bound on the empirical error or *training error* of the hypothesis output by AdaBoost as a function of the number of rounds T and the value of the advantage γ of the weak learner used. We will see that the training error goes to 0 extremely fast. In the next lecture, we will see what this implies about test error and generalization.

Let γ_t be a measure of how much better than random guessing h_t is with respect to $\mathcal{D}_t$, defined as

$$\gamma_t = \frac{1}{2} - \epsilon_t = \frac{1}{2} - \Pr_{\vec{x} \sim \mathcal{D}_t}(h_t(\vec{x}) \neq c(\vec{x})).$$

We will prove the following result.

Theorem 1. *Let h be the function output by AdaBoost after T rounds, and let γ_t be as defined above. Then*

$$\widehat{err}(h) \leq e^{-2\sum_{t=1}^{T} \gamma_t^2} .$$

Thus if $\gamma_t \geq \gamma$ for all t (by the weak learning assumption), then

$$\widehat{err}(h) \leq e^{-2\gamma^2 T}.$$

This theorem gives a bound on the training error of the hypothesis output by AdaBoost that decreases exponentially fast as the number of rounds T grows. Since the training error is always a multiple of $1/m$, and the bound on training error becomes lower than $1/m$ quickly as T increases, this implies that AdaBoost attains a training error of zero (i.e., consistency) in very few rounds.

Notice that while the error bound depends on γ, the algorithm itself does not. Hence we get this guarantee without having to know γ in advance.

3.1 Proof of Theorem 1

The proof of this bound will proceed in three steps:

1. Unravel the recurrence on $\mathcal{D}_t$ to show that
$$\mathcal{D}_{T+1}(i) = \frac{e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)}}{m \prod_{t=1}^{T} Z_t} \ .$$
2. Use Step 1 to show that
$$\widehat{\text{err}}(h) \leq \prod_{t=1}^{T} Z_t \ .$$
3. Show that after optimizing α_t, $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$ for all t, and plug this into Step 2 to achieve the bound.

Step 1

We unravel the recurrence for $\mathcal{D}_t$ to get

$$\begin{aligned}
\mathcal{D}_{T+1}(i) &= \frac{\mathcal{D}_T(i) e^{-\alpha_T y_i h_T(\vec{x}_i)}}{Z_T} \\
&= \frac{\mathcal{D}_{T-1}(i) e^{-\alpha_{T-1} y_i h_{T-1}(\vec{x}_i)} e^{-\alpha_T y_i h_T(\vec{x}_i)}}{Z_T Z_{T-1}} \\
&\quad \text{(continue recurrence substitutions until we reach } \mathcal{D}_1(i)) \\
&= \mathcal{D}_1(i) \frac{\prod_{t=1}^{T} e^{-\alpha_t y_i h_t(\vec{x}_i)}}{\prod_{t=1}^{T} Z_t} \\
&= \frac{\prod_{t=1}^{T} e^{-\alpha_t y_i h_t(\vec{x}_i)}}{m \prod_{t=1}^{T} Z_t} \\
&= \frac{e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(\vec{x}_i)}}{m \prod_{t=1}^{T} Z_t} \\
&= \frac{e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)}}{m \prod_{t=1}^{T} Z_t}.
\end{aligned}$$

Step 2

Now we try to bound the empirical error $\widehat{\text{err}}(h)$. To do this, we first consider the exponential term

$$e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)}.$$

If the final classifier makes a mistake on the point $\vec{x}_i$, then

$$\text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)\right) \neq \text{y}_i$$

and so

$$\text{sign}\left(y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)\right) = -1$$

and $-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i) \geq 0$. Hence

$$e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)} \geq 1 \quad \text{if} \quad h(\vec{x}_i) \neq y_i.$$

Also, since $e^x \geq 0 \; \forall x$,

$$e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)} \geq 0 \quad \text{if} \quad h(\vec{x}_i) = y_i.$$

Using these facts and the result from Step 1, we get

$$\begin{aligned}
\widehat{\text{err}}(h) &= \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}(h(\vec{x}_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{T} \alpha_t h_t(\vec{x}_i)} \\
&= \frac{1}{m} \sum_{i=1}^{m} m\mathcal{D}_{T+1}(i) \prod_{t=1}^{T} Z_t \\
&= \sum_{i=1}^{m} \mathcal{D}_{T+1}(i) \prod_{t=1}^{T} Z_t \\
&= \prod_{t=1}^{T} Z_t.
\end{aligned}$$

This gives $\widehat{\text{err}}(h) \leq \prod_{t=1}^{T} Z_t$.

Step 3

Finally, we bound the values of Z_t. We first take the definition of Z_t and break up the sum into 2 parts: a sum over all points labeled correctly and a sum over all points labeled incorrectly.

$$\begin{aligned}
Z_t &= \sum_{i=1}^{m} \mathcal{D}_t(i) e^{-\alpha_t y_i h_t(\vec{x}_i)} \\
&= \sum_{i:\; y_i = h_t(\vec{x}_i)} \mathcal{D}_t(i) e^{-\alpha_t} + \sum_{i:\; y_i \neq h_t(\vec{x}_i)} \mathcal{D}_t(i) e^{\alpha_t} \\
&= e^{-\alpha_t} \sum_{i:\; y_i = h_t(\vec{x}_i)} \mathcal{D}_t(i) + e^{\alpha_t} \sum_{i:\; y_i \neq h_t(\vec{x}_i)} \mathcal{D}_t(i) \\
&= (1 - \epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}
\end{aligned}$$

As a brief aside, we can verify at this point why the particular choice of α_t used in the algorithm is good. We know from Step 2 that our bound will be minimized when we make each Z_t as small as possible. We can solve for the value of α_t that minimizes Z_t for each t. Setting the derivative of Z_t equal to 0, we obtain

$$0 = \frac{dZ_t}{d\alpha_t} = -(1 - \epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t}.$$

Solving for α_t gives us the optimal values of the parameters,

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) .$$

Note that it's ok to let α_t depend on ϵ_t because ϵ_t can be calculated from h_t alone and the data at the time when we need it.

Plugging this value of α_t into the expression for Z_t above, we get

$$Z_t = (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 2\sqrt{(1-\epsilon_t)\epsilon_t}.$$

Putting It All Together

Now combining Steps 2 and 3, we get

$$\begin{aligned}
\widehat{\text{err}}(h) &\le \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{(1-\epsilon_t)\epsilon_t} \\
&\quad (\text{using } \; \gamma_t = \frac{1}{2} - \epsilon_t) \\
&= \prod_{t=1}^{T} 2\sqrt{\left(\frac{1}{2} - \gamma_t\right)\left(\frac{1}{2} + \gamma_t\right)} \\
&= \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \\
&\quad (\text{using } \; 1 + x \le e^x) \\
&\le \prod_{t=1}^{T} \sqrt{e^{-4\gamma_t^2}} = \prod_{t=1}^{T} e^{-2\gamma_t^2} = e^{-2\sum_{t=1}^{T} \gamma_t^2} \le e^{-2T\gamma^2}.
\end{aligned}$$

This completes the proof. □

We will discuss the generalization error of AdaBoost in the next lecture.