Disclaimer: These notes are designed to be a supplement to the lecture. They may or may not cover all the material discussed in the lecture (and vice versa).

Outline

• Introduction
• AdaBoost algorithm
• Analysis
• Loss minimization view

1 Introduction

In this lecture we come back to batch (supervised) learning, and consider the question of converting a ‘weak’ learning algorithm into a ‘strong’ learning algorithm. In particular, suppose we have some ‘weak’ learning algorithm, or just some collection of ‘rules of thumb’, which given any finite number of labeled training examples together with some probability distribution or weights on them, returns a classifier (some rule of thumb) with just slightly better than 50% accuracy on that distribution (i.e. just slightly better than random guessing). For example, in an email classification problem, we might have rules of thumb like {if an email contains the word “free”, classify it as spam}, {if an email contains the word “seminar”, classify it as non-spam}, etc; given any weighted set of examples, we may be able to find a rule of thumb that gives at least 50% accuracy on them. Can we somehow use such a weak learning algorithm and turn it into a ‘strong’ learning algorithm, which can yield classifiers with higher accuracy?

We will see how this can be done using the AdaBoost algorithm. We focus on binary classification, although the algorithm can be extended to many other supervised learning algorithms.

2 AdaBoost Algorithm

Assume a weak learning algorithm that given as input a weighted training sample \((S, D)\), where \(S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{\pm 1\})^m\) and \(D\) is a probability distribution over the examples in \(S\), returns a classifier \(h: \mathcal{X} \rightarrow \{\pm 1\}\) that has slightly better accuracy than random guessing on the training sample \((S, D)\). In particular, the error on the weighted sample \((S, D)\) will be measured by

\[
er_D[h] = \sum_{i=1}^m D(i) \cdot 1(h(x_i) \neq y_i),
\]
where \( D(i) \) is the weight on example \((x_i, y_i)\). (Note that if \( D(i) = \frac{1}{m} \) for all \( i \), then \( \mathrm{er}_D[h] \) is simply the usual training error.)

The AdaBoost algorithm works in rounds \( t \), and maintains a probability distribution (set of weights) \( D_t \) over the training examples in \( S \); the initial distribution \( D_1 \) is simply the uniform distribution over all \( m \) examples in \( S \). On each round \( t \), the algorithm gives the weighted training sample \((S, D_t)\) to the weak learner, and gets a classifier \( h_t \). We will denote by \( \mathrm{er}_t \) the error of \( h_t \) on the weighted sample \((S, D_t)\):

\[
\mathrm{er}_t = \mathrm{er}_{D_t}[h_t].
\]

The central idea is that on each round \( t \), the algorithm updates the distribution \( D_t \) on the training examples such that in the next round, the weak learner is forced to pay more attention to those examples that are not classified well by \( h_t \) (as we will see, this has the effect of boosting the accuracy of the final learned classifier). The final classifier output by the algorithm is a weighted majority vote of the weak classifiers \( h_t \).

The algorithm is summarized below:

---

**Algorithm AdaBoost**

**Inputs:** Training sample \( S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{\pm 1\})^m \)

Number of iterations \( T \)

**Initialize:** \( D_1(i) = \frac{1}{m} \forall i \in [m] \)

For \( t = 1, \ldots, T \):
- Train weak learner on weighted sample \((S, D_t)\); get weak classifier \( h_t : \mathcal{X} \to \{\pm 1\} \)
- Set \( \alpha_t \leftarrow \frac{1}{2} \ln \left( \frac{1 - \mathrm{er}_t}{\mathrm{er}_t} \right) \)
- Update:
  \[
  D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}
  \]
  where \( Z_t = \sum_{j=1}^{m} D_t(j) \exp(-\alpha_t y_j h_t(x_j)) \)

**Output final hypothesis:**

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)
\]

---

### 3 Analysis

In the following, for each round \( t \), we will denote by \( F_t : \mathcal{X} \to \mathbb{R} \) the partial combination of weak classifiers constructed so far:

\[
F_t(x) = \sum_{s=1}^{t} \alpha_s h_s(x).
\]

In order to understand the algorithm, we first observe that on each round \( t \), the distribution \( D_{t+1} \) weights examples in accordance with the ‘margins’ of the partial (real-valued) classifier \( F_t \):

**Lemma 1.** For each round \( t \):

\[
D_{t+1}(i) \propto e^{-y_i F_t(x_i)}.
\]
Proof. We have,

\[
D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad (1)
\]

\[
= \frac{D_{t-1}(i) \exp(-\alpha_{t-1} y_i h_{t-1}(x_i)) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_{t-1} \cdot Z_t} \quad (2)
\]

\[
\vdots
\]

\[
= \frac{D_t(i) \exp(-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i))}{\prod_{s=1}^{t} Z_s} \quad (3)
\]

\[
= \frac{\frac{1}{m} \exp(-y_i F_t(x_i))}{\prod_{s=1}^{T} Z_s} \quad (4)
\]

Thus the distribution \(D_{t+1}\) assigns weights to examples \((x_i, y_i)\) in \(S\) according to the margins \(y_i F_t(x_i)\) of the function \(F_t\) obtained so far: the smaller the margin (the poorer the classification), the larger the weight. This forces the weak learner on the \((t+1)\)-th round to focus on examples not classified accurately by \(F_t\).

Next, let us denote the training error of the final classifier \(H\) by \(\text{er}_S[H]\):

\[
\text{er}_S[H] = \frac{1}{m} \sum_{i=1}^{m} 1(H(x_i) \neq y_i).
\]

We make the following observation:

**Lemma 2.** \(\text{er}_S[H] \leq \frac{1}{m} \sum_{i=1}^{m} e^{-y_i F_T(x_i)} = \prod_{s=1}^{T} Z_s\).

*Proof.* The inequality follows since for each \(i\), we have

\[
1(H(x_i) \neq y_i) = 1(y_i F_T(x_i) < 0) \leq e^{-y_i F_T(x_i)}.
\]

The equality follows by applying Eq. (4) to \(t = T\), summing over all \(i\), and noting that \(\sum_{i=1}^{m} D_{T+1}(i) = 1\). \(\square\)

We also note that with the particular choice of \(\alpha_t\) made by the algorithm, the normalization constant \(Z_t\) has a simple form:

**Lemma 3.** With the given choice of \(\alpha_t\), for each round \(t\): \(Z_t = 2 \sqrt{\text{er}_t(1 - \text{er}_t)}\).

*Proof.* First note that

\[
e^{-\alpha_t} = \sqrt{\frac{\text{er}_t}{1 - \text{er}_t}}
\]

and

\[
y_i h_t(x_i) = \begin{cases} +1 & \text{if } h_t(x_i) = y_i \\ -1 & \text{otherwise.} \end{cases}
\]
Now, we have

\[
Z_t = \sum_{i=1}^{m} D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}
\]

(5)

\[
= \sum_{i=1}^{m} D_t(i) \left( e^{-\alpha_t} \mathbf{1}(h_t(x_i) = y_i) + e^{\alpha_t} \mathbf{1}(h_t(x_i) \neq y_i) \right)
\]

(6)

\[
= e^{-\alpha_t} \sum_{i=1}^{m} D_t(i) \cdot \mathbf{1}(h_t(x_i) = y_i) + e^{\alpha_t} \sum_{i=1}^{m} D_t(i) \cdot \mathbf{1}(h_t(x_i) \neq y_i)
\]

(7)

\[
= \sqrt{\frac{e r_t}{1 - e r_t}} (1 - e r_t) + \sqrt{\frac{1 - e r_t}{e r_t}} e r_t
\]

(8)

\[
= 2 \sqrt{e r_t (1 - e r_t)}.
\]

(9)

We are now ready for the following main result, which shows that if the weak classifiers \( h_t \) returned by the weak learner all have accuracies slightly better than random guessing, then for large enough \( T \), the training error of the final classifier \( H \) can be made as small as desired:

**Theorem 4.** Let \( \gamma \in (0, \frac{1}{2}] \). If \( e r_t \leq \frac{1}{2} - \gamma \) for all \( t \), then \( e r_S[H] \leq e^{-2T\gamma^2} \).

**Proof.** Suppose \( e r_t \leq \frac{1}{2} - \gamma \) for all \( t \). Then we have,

\[
\begin{align*}
er_S[H] & \leq \prod_{t=1}^{T} Z_t, \quad \text{by Lemma 2} \\
& = 2^T \prod_{t=1}^{T} \sqrt{e r_t (1 - e r_t)}, \quad \text{by Lemma 3} \\
& \leq 2^T \prod_{t=1}^{T} \sqrt{\left(\frac{1}{2} - \gamma\right)\left(\frac{1}{2} + \gamma\right)}, \\
& \quad \text{since } e r_t \leq \frac{1}{2} - \gamma \text{ and since } \sqrt{p(1-p)} \text{ is an increasing function of } p \text{ on } [0, \frac{1}{2}] \\
& = 2^T \prod_{t=1}^{T} \sqrt{\frac{1}{4} - \gamma^2} \\
& = (1 - 4\gamma^2)^{T/2} \\
& \leq e^{-2T\gamma^2}, \quad \text{since } 1 - x \leq e^{-x}.
\end{align*}
\]

(10) (11) (12) (13) (14) (15) (16)

One can also allow the weak classifiers to produce real-valued classifiers \( f_t : \mathcal{X} \to \mathbb{R} \) instead of binary classifiers \( h_t : \mathcal{X} \to \mathbb{R} \); in that case the definition of error for the weak classifiers \( f_t \) changes somewhat, and one needs to choose \( \alpha_t \) differently. In general, \( \alpha_t \) is chosen to minimize the resulting \( Z_t \); in the case of binary-valued weak classifiers, the above choice of \( \alpha_t \) minimizes \( Z_t \) exactly (exercise!), but more generally, one may need to choose \( \alpha_t \) to approximately minimize \( Z_t \).
4 Loss Minimization View

Consider the exponential loss function $\ell_{\exp} : \{\pm 1\} \times \mathbb{R} \rightarrow \mathbb{R}_+$ defined as

$$\ell_{\exp}(y, f) = e^{-yf}.$$ 

Then as we saw in Lemma 2, the average exponential loss of the (real-valued) classifier $F_T$ learned by AdaBoost on the training sample $S$ is equal to the product of the normalization factors $Z_t$ over the $T$ rounds:

$$\frac{1}{m} \sum_{i=1}^{m} \ell_{\exp}(y_i, F_T(x_i)) = \prod_{t=1}^{T} Z_t.$$ 

As noted above, the parameter $\alpha_t$ is chosen to minimize $Z_t$ on each round $t$. Thus AdaBoost can also be viewed as a greedy/coordinate descent style approach to minimize the average exponential loss on the training sample (over functions consisting of linear combinations of weak classifiers)!