CSE 252C: Computer Vision III

Lecturer: Serge Belongie
Scribes: Andrew Rabinovich and Vincent Rabaud
Edited by: Catherine Wah

LECTURE 16
Boosting

16.1. Introduction

Since Viola & Jones (2001), boosting – and AdaBoost (Freund & Schapire, 1997) – has taken on a prominent role in computer vision, in particular for object detection. The idea behind boosting is to combine the outputs of many "weak learners" in a weighted fashion to produce a "strong" classifier. We'll focus on the 2-class case in this lecture; a generalization to multiclass exists.

Definition 16.1. A *weak learner* is a classifier that is correct at least a bit more than half of the time, *i.e.* it is slightly better than chance.

Formally, suppose we are given a training set:

$$D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\} \text{ with } \boldsymbol{x}_i \in \mathbb{R}^D, y = \{-1, 1\}\,.$$

[1]Department of Computer Science and Engineering, University of California, San Diego.

October 11, 2009

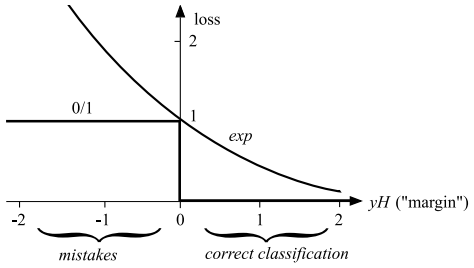

Figure 1. Exponential loss is the upper bound on the 0/1 loss.

A weak learner is a function $h_k : \boldsymbol{x} \mapsto y$, and the strong classifier has the form:

$$H(\boldsymbol{x}) = \sum_{k=1}^{T} \alpha_k h_k(\boldsymbol{x}). \tag{16.2}$$

The problem is how to set the weights α_k and train the weak learners $h_k(\boldsymbol{x})$. We'll examine now how this is done with AdaBoost, specifically, discrete AdaBoost (Adaptive Boosting).

16.2. AdaBoost

Ideally, one would like to minimize what's known as the 0/1 loss, *i.e.* a penalty of 1 for incorrect classification, and no penalty for correct classification. A 0/1 loss is not differentiable, however, which makes it difficult to work with. AdaBoost uses the so-called exponential loss (see Figure 1):

$$\mathcal{L}(H) = \sum_{i=1}^{n} e^{-y_i H(\boldsymbol{x}_i)}. \tag{16.3}$$

Observe that $y_i \in \{-1, 1\}$, and $H(\boldsymbol{x}_i)$ is multiplied by this; thus, we want it to be the same sign as y_i.

Now, suppose we are given the first t weak classifiers, and we want to specify h_{t+1} and α_{t+1}. Let $H_t(\boldsymbol{x}) = \sum_{k=1}^{t} \alpha_k h_k(\boldsymbol{x})$ denote the strong classifier, made up of the first t weak learners. Iteration $t+1$ can then be posed as follows:

$$(h_{t+1}, \alpha_{t+1}) = \arg\min_{h,\alpha} \mathcal{L}(H_t + \alpha h) \tag{16.4}$$

$$= \arg\min_{h,\alpha} \sum_{i=1}^{n} e^{-y_i(H_t(\boldsymbol{x}_i) + \alpha h(\boldsymbol{x}_i))} \tag{16.5}$$

$$= \arg\min_{h,\alpha} \sum_{i=1}^{n} w_t(i) e^{-y_i \alpha h(\boldsymbol{x}_i)} \tag{16.6}$$

where $w_t(i) = \exp(-y_i H_t(\boldsymbol{x}_i))$ doesn't depend on the min arguments; we can think of it as a per-point weighting at iteration t.

Furthermore, we can assume the weights have been normalized to sum to 1, w.l.o.g.: $\sum_{i=1}^n w_t(i) = 1$. (The $\arg\min$ is not affected.) Now, let's do some more manipulations:

$$(h_{t+1}, \alpha_{t+1}) = \arg\min_{h,\alpha} \sum_{i=1}^n w_t(i) \exp(-y_i \alpha h(\boldsymbol{x}_i)) \tag{16.7}$$

and break this into two cases:

$$\begin{aligned}
(h_{t+1}, \alpha_{t+1}) &= \arg\min_{h,\alpha} e^{-\alpha} \sum_{i|y_i = h(\boldsymbol{x}_i)} w_t(i) + e^{\alpha} \sum_{i|y_i \neq h(\boldsymbol{x}_i)} w_t(i) \\
&= \arg\min_{h,\alpha} (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^n w_t(i) \mathbf{1}[y_i \neq h(\boldsymbol{x}_i)] + e^{-\alpha} \underbrace{\sum_{i=1}^n w_t(i)}_{1} \\
&= \arg\min_{h,\alpha} \underbrace{(e^{\alpha} - e^{-\alpha})}_{\text{pos.constant}} \sum_{i=1}^n w_t(i) \mathbf{1}[y_i \neq h(\boldsymbol{x}_i)] + \underbrace{e^{-\alpha}}_{\text{pos. constant}}.
\end{aligned}$$

Therefore:

$$h_{t+1} = \arg\min_{h} \sum_{i=1}^n w_t(i) \mathbf{1}[y_i \neq h(\boldsymbol{x}_i)]. \tag{16.8}$$

Note how the errors are weighted differently per sample.

How about α_{t+1}? We define

$$\epsilon_t = \sum_{i=1}^n w_t(i) \mathbf{1}[y_i \neq h(\boldsymbol{x}_i)], \tag{16.9}$$

one weighted error of the new weak classifier. We need to solve $\alpha_{t+1} = \arg\min_{\alpha}(e^{\alpha} - e^{-\alpha})\epsilon_t + e^{-\alpha}$. Taking the derivative and setting it to zero we get:

$$\begin{aligned}
\frac{\partial}{\partial \alpha} \left((e^{\alpha} - e^{-\alpha})\epsilon_t + e^{-\alpha} \right) = e^{\alpha}\epsilon_t + e^{-\alpha}\epsilon_t - e^{-\alpha} &= 0 \\
e^{2\alpha}\epsilon_t + \epsilon_t - 1 &= 0 \\
e^{2\alpha} &= \frac{1 - \epsilon_t}{\epsilon_t}.
\end{aligned}$$

Hence:

$$\alpha_{t+1} = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}. \tag{16.10}$$

Now we're close to being able to state the AdaBoost algorithm, or more accurately, meta-algorithm. Recall we normalized the weights, which we could denote:

$$w_t(i) = \frac{1}{Z_t} \exp\left(-y_i H_t(\boldsymbol{x}_i)\right),$$

where Z_t is the normalization factor. On the next iteration, we could set the weights as:

$$w_{t+1}(i) = \frac{1}{Z_{T+1}} \exp\left(-y_i (H_t(\boldsymbol{x}_i) + \alpha_{t+1} h_{t+1}(\boldsymbol{x}_i))\right)$$

and normalize again, but we can do more simplification. Notice the redundancy:

$$\begin{aligned} w_{t+1}(i) &= \frac{1}{Z_{t+1}} \exp\left(-y_i (H_t(\boldsymbol{x}_i) + \alpha_{t+1} h_{t+1}(\boldsymbol{x}_i))\right) \\ &= w_t(i) \frac{Z_t}{Z_{t+1}} \exp\left(-y_i \alpha_{t+1} h_{t+1}(\boldsymbol{x}_i)\right). \end{aligned}$$

Now we use a simple trick:

$$\begin{aligned} -y h(\boldsymbol{x}_i) &= 2 \cdot \mathbf{1}[y_i \neq h(\boldsymbol{x}_i)] - 1 \\ \Rightarrow w_{t+1}(i) &= w_t(i) \frac{Z_t}{Z_{t+1}} \exp\left(2\alpha_{t+1} \mathbf{1}[y_i \neq h_{t+1}(\boldsymbol{x}_i)]\right) e^{-\alpha}. \end{aligned}$$

Note that Z_t and $e^{-\alpha}$ are constants, and they don't affect the above arg min, so we can just absorb them into Z_{t+1}:

$$\begin{aligned} w_{t+1}(i) &= w_t(i) \frac{1}{Z_{t+1}} \exp\left(2\alpha_{t+1} \mathbf{1}[y_i \neq h_{t+1}(\boldsymbol{x}_i)]\right) \\ &= w_t(i) \frac{1}{Z_{t+1}} \exp\left(\log \frac{1-\epsilon_t}{\epsilon_t} \mathbf{1}[y_i \neq h_{t+1}(\boldsymbol{x}_i)]\right). \end{aligned}$$

Finally, observe that:

$$\begin{aligned} Z_{t+1} &= \sum_{i | y_i \neq h_{t+1}(\boldsymbol{x}_i)} w_t(i) \frac{1-\epsilon_t}{\epsilon_t} + \sum_{i | y_i = h_{t+1}(\boldsymbol{x}_i)} w_t(i) \\ &= \epsilon_t \frac{1-\epsilon_t}{\epsilon_t} + 1 - \epsilon_t \\ &= 2(1-\epsilon_t), \end{aligned}$$

which leads to a simple expression for setting the value of the new weight:

$$w_{t+1}(i) = \begin{cases} \frac{w_t(i)}{2(1-\epsilon_t)} & \text{if } y_i = h_{t+1}(\boldsymbol{x}_i) \\ \frac{w_t(i)}{2\epsilon_t} & \text{otherwise.} \end{cases} \tag{16.11}$$

16.2.1. Discrete AdaBoost

Now we can state the Discrete AdaBoost algorithm (Schapire, 1997) (Algorithm 16.1). The empirical error is upper-bounded by $\prod_{t=1}^{T} Z_t$ (exercise for the reader: why is this the case?).

Algorithm 16.1 Discrete AdaBoost

Input: Training data $\mathcal{D}$, number of iterations T, initial distribution $w_0(i)$ (*e.g.* uniform) over data points
Output: A strong classifier $H(\boldsymbol{x})$
for $t = 1, \ldots, T$ **do**
  train a weak classifier: $h_t = \arg\min_h \sum_{i=1}^{n} w_{t-1}(i)\mathbf{1}[y_i \neq h(\boldsymbol{x}_i)]$
  calculate error: $\epsilon_t = \sum_{i=1}^{n} w_{t-1}(i)\mathbf{1}[y_i \neq h_t(\boldsymbol{x}_i)]$
  set: $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  set: $w_{t+1}(i) = \begin{cases} \frac{w_t(i)}{2(1-\epsilon_t)} & \text{if } y_i = h_{t+1}(\boldsymbol{x}_i) \\ \frac{w_t(i)}{2\epsilon_t} & \text{otherwise} \end{cases}$
end for
return $H(\boldsymbol{x}) = \sum_{k=1}^{T} \alpha_k h_k(\boldsymbol{x})$

16.2.2. Discussion

Some of the pros of AdaBoost are that it is very simple to implement, it supports feature selection on very large sets of features, and has fairly good generalization. However, it can overfit in presence of noise and outliers, and yields a suboptimal solution for α_t's.

Many variants of AdaBoost have been proposed (*e.g.* LogitBoost, GentleBoost, BrownBoost, ...); they are not covered in this class. These address some of the cons listed above.

A notable example of AdaBoost applied to the problem of object detection is Viola-Jones, which combined AdaBoost, cascade, and Haar-like filters.