CIS5200: Machine Learning Spring 2023

Lecture 19: Boosting

Date: March 28, 2023 *Author: Surbhi Goel*

Acknowledgements. These notes are heavily inspired by notes by Rob Schapire (Princeton) and Kilian Weinberger (Cornell).

Disclaimer. These notes have not been subjected to the usual scrutiny reserved for formal publications. If you notice any typos or errors, please reach out to the author.

1 Boosting

Last class we looked at a procedure called Bagging that helps reduce variance of the model. In this class we will look at a procedure called boosting that helps reduce bias of the model. Boosting is perhaps the most successful paradigm to emerge out of learning theory that is very actively used even today. The core idea here is to *boost* simple learners that do a bit better than random guessing and combine them in a way to get a more complicated learner that performs very well. The question of whether this is possible was in fact posed by Michael Kearns in a machine learning class project in 1988, and later answered in affirmative by Rob Schaphire in 1990.

Let us formalize the setup:

Data: Training set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ where $y_i \in \{-1, 1\}$

Weak Learner: *Weak* learning algorithm $\mathcal{A}$ that guarantees better than random performance, for example, is 55% of the time correct. More formally, given a function class $\mathcal{F}$, for any distribution $\mathcal{D}$ on inputs and labelling function $f \in \mathcal{F}$, given a large enough training set drawn from $\mathcal{D}$ and labelled according to f, with probability $1 - \delta$, the algorithm produces a hypothesis $\hat{f}$ that has non-trivial risk $\mathcal{R}(\hat{f}) \leq 1/2 - \gamma$. This is the same definition as PAC learning with $\epsilon = 1/2 - \gamma$.

Goal: Output a classifier f that gets error ϵ using the weak learner as a sub-routine. f need not be a function in $\mathcal{F}$.

2 Generic Boosting Algorithm

A simple way to use the weak learner would be to run it multiple times on the dataset and combine the prediction using perhaps the majority vote. However, if $\mathcal{A}$ is deterministic, then re-running it multiple times will not help get a better estimate. This suggests that we need to modify the data before re-running $\mathcal{A}$.

One way to modify the data is re-weight it. Let us formalize this by considering a discrete distribution $\mu = (\mu_1, \ldots, \mu_m)$ over $[m]$. Using this, we get a generic strategy:

Algorithm 1: Generic Boosting Scheme

for $t = 1, 2, \ldots, T$ **do**
  Construct discrete distribution μ_t over $[m]$
  Run weak learner $\mathcal{A}$ on training data sampled according to μ_t to get classifier f_t with small error over μ_t, $\epsilon_t = \Pr_{i\sim\mu_t}[f_t(x_i) \neq y_i] = 1/2 - \gamma_t \leq 1/2 - \gamma$ (by weak learning assumption).
end
Output final classifier f constructed using $f_1, \ldots, f_T$.

This brings us to two questions: (1) How do we choose μ_t? (2) How do we construct final classifier f using $f_1, \ldots, f_T$?

3 AdaBoost

AdaBoost or Adaptive Boosting (Freund and Schapire 98) constructs μ_t by increasing the weights on *hard* examples that the weak learner makes mistakes on, and decreasing weights on *easy* examples that the weak learner gets right. In particular, for $i \in [m]$

$$\begin{aligned} \mu_{1,i} &= \frac{1}{m} \\ \mu_{t+1,i} &= \frac{\mu_{t,i}}{Z_t} \times \exp(-\alpha_t y_i f_t(x_i)), \end{aligned}$$

where $\alpha_t > 0$ is the penalty, and $Z_t = \sum_{j=1}^{m} \mu_{t,j} \exp(-\alpha_t y_j f_t(x_j))$ is the normalizing constant that ensures μ_{t+1} is a valid distribution. Observe that the easy examples ($y_i = f_t(x_i)$) are down-weighted and hard examples ($y_i \neq f_t(x_i)$) are up-weighted.

The final classifier is a weighted combination of the weak learners:

$$f(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(x)\right).$$

The optimal choice of parameter α_t is:

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

AdaBoost is one of the most widely used algorithms because of its ease of implementation. It does not require knowing γ, the weak learning parameter, and is adaptive to whatever value γ has. In fact, we can get strong guarantees for the performance of the final classifier f output by AdaBoost.

Theorem 1. *Let f be the output of AdaBoost after T steps, then we have*

$$\hat{R}(f) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}[f(x_i) \neq y_i] \leq \exp\left(-2\gamma^2 T\right).$$

Note that the above theorem shows that the the training error goes down exponentially with the number of iterations.

Proof. We will prove this in 3 steps:

Bound on μ_{T+1}: We have

$$\begin{aligned}
\mu_{T+1,i} &= \frac{\mu_{T,i}}{Z_T} \times \exp(-\alpha_T y_i f_T(x_i)) && \textit{Applying the weight update formula} \\
&= \frac{1}{m} \times \cdots \times \frac{\exp(-\alpha_{T-1} y_i f_{T-1}(x_i))}{Z_{T-1}} \times \frac{\exp(-\alpha_T y_i f_T(x_i))}{Z_T} && \textit{Expanding from iteration 1 to } T \\
&= \frac{1}{m} \prod_{t=1}^{T} \frac{\exp(-\alpha_t y_i f_t(x_i))}{Z_t} && \textit{Combining the expression} \\
&= \frac{\exp\left(-\sum_{t=1}^{T} \alpha_t y_i f_t(x_i)\right)}{m \prod_{t=1}^{T} Z_t} && \textit{Factoring out the exponent} \\
&= \frac{\exp\left(-y_i f(x_i)\right)}{m \prod_{t=1}^{T} Z_t} && \textit{Substituting the weighted classifier as } f
\end{aligned}$$

Bound on $\hat{R}(f)$: Observe that

$$\begin{aligned}
\hat{R}(f) &= \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[f(x_i) \neq y_i] && \textit{Definition of training error} \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[y_i f(x_i) \leq 0] && \textit{Rewriting using the inequality } y_i f(x_i) <= 0 \\
&\leq \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i f(x_i)) && \textit{Upper bounding the indicator by the exponential} \\
&= \left(\prod_{t=1}^{T} Z_t\right) \times \sum_{i=1}^{m} \mu_{T+1,i} && \textit{Substituting the bound on } \mu_{T+1} \textit{ from before} \\
&= \prod_{t=1}^{T} Z_t && \textit{Simplifying the expression}
\end{aligned}$$

Bound on Z_t: We have,

$$\begin{aligned}
Z_t &= \sum_{j=1}^{m} \mu_{t,j} \exp(-\alpha_t y_j f_t(x_j)) && \textit{Definition of } Z_t \\
&= \sum_{j=1}^{m} \mu_{t,j} \left(\mathbb{1}[y_j = f_t(x_j)] e^{-\alpha_t} + \mathbb{1}[y_j \neq f_t(x_j)] e^{\alpha_t}\right) && \textit{Expanding the exponential using indicators} \\
&= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} && \textit{Observing that } \sum_{j=1}^{m} \mu_{t,j} \mathbb{1}[y_j \neq f_t(x_j)] \textit{ is } \epsilon_t \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)} && \textit{Substituting the value of } \alpha_t \\
&= \sqrt{1-4\gamma_t^2} && \textit{Substituting the value of } \gamma_t \\
&\leq \exp(-2\gamma_t^2) && \textit{Using } 1 + a \leq \exp(a) \\
&\leq \exp(-2\gamma^2) && \textit{Using } \gamma_t \geq \gamma
\end{aligned}$$

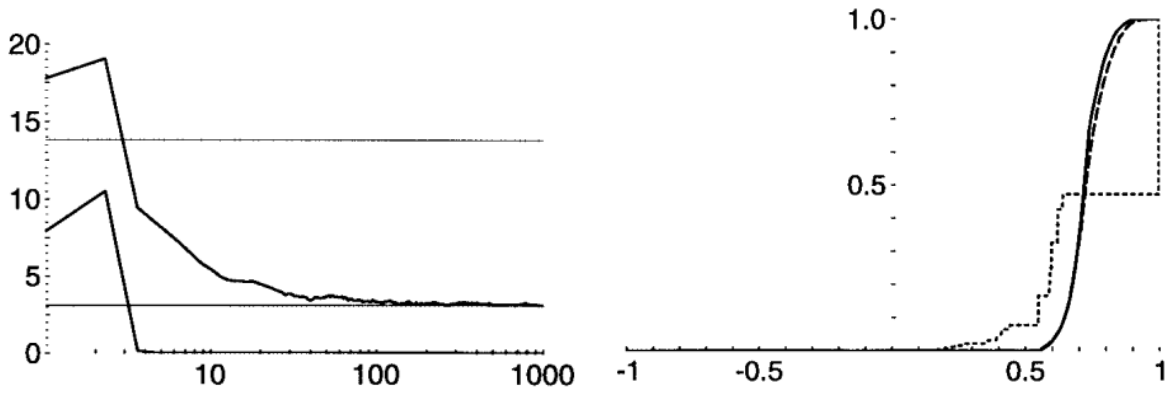

Figure 1: From Schapire et al'98 showing that boosting does not overfit, due to maximizing margin. (*left*) x-axis is number of rounds, y-axis is error (in %). The top curve is test error and the lower curve is training error. (*right*) x-axis is margin, y-axis is cumulative distribution. The three curves correspond to different training iterations.

Now combining the above two, we have

$$\hat{R}(f) \leq \prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} \exp(-2\gamma^2) = \exp(-2\gamma^2 T).$$

□

4 Generalization Performance of AdaBoost

In the previous section we showed that the training error goes down exponentially fast. It also seems that the classifier we get with more iterations gets more complex. This would imply that even though we reduce bias, we are increasing variance, leading to overfitting. However, more often than not, we do not observe any overfitting as T gets larger. See Figure 1 (left) for an example of a setup where no overfitting is observed. In fact, test error continues to drop even after training error has stopped improving.

Why does this happen? Well, the answer is *margins*. Much like we saw in SVM, here the boosting algorithm implicitly ensures large margin under a slightly different notion of margin. Here the margin is based on the confidence in prediction, that is, weight of correct classifiers minus the weight of incorrect classifiers. If this quantity is large, we are pretty sure of our prediction. If it is negative, our prediction is incorrect.Observe that in Figure 1 (right), the margin for all points is at least 0.5. This allows for generalization.

Note: This phenomenon is also observed in neural networks where overfitting does not imply poor generalization.

5 AdaBoost as a Coordinate Descent Algorithm

AdaBoost can be reinterpreted as a coordinate descent algorithm operating within the function space of weak classifiers. From this standpoint, the optimization problem is formulated as the minimization of the exponential loss over a function f, which is a linear combination of weak classifiers $f_1, \ldots$.

The exponential loss function $\ell(\alpha)$ is expressed as follows:

$$\ell(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i f(x_i)) = \frac{1}{m} \sum_{i=1}^{m} \exp\left(-y_i \sum_{j=1}^{\infty} \alpha_j f_j(x_i) \right).$$

Within the context of coordinate descent, the optimization process entails selecting a single coordinate and minimizing the loss function concerning that coordinate. Given that the number of weak hypothesis is infinite, we need a process for coordinate selection. This is done by identifying the weak classifier f_t that yields the largest decrease in loss (steepest descent) and subsequently determining the corresponding α_t to take in that direction.

Remarkably, this procedure exhibits strong parallels with the AdaBoost algorithm. We will skip the details of how exactly these two match.