601.433/633 Introduction to Algorithms **Lecturer:** Michael Dinitz and Jessica Sorrell
Topic: Multiplicative Weights **Date:** 12/4/25

25.1 Introduction

Last time, we introduced the online learning problem, the notion of no-regret algorithms, and showed that the multiplicative weights algorithm is one such no-regret algorithm. Today we'll continue our discussion of the multiplicative weights algorithm and its applications to problems outside of online learning. In particular, we will focus on boosting algorithms. Boosting algorithms are a family of algorithms which "boost" a weak learner – a learner that does only a little bit better than random guessing – to one that is almost always correct.

While there are boosting algorithms for the online learning setting we discussed last time, today we will focus on boosting algorithms for another common learning setting known as *supervised learning.*

25.2 Supervised Learning

The following basic problem arises often in machine learning: there is some unknown set, and we are given a sample of some elements and told whether or not each element is in the set (i.e. we are given *labeled* data). From this data, we want to produce a good prediction rule (a *hypothesis*) for future data.

A standard example of this problem is spam categorization. We want a computer program to help us decide which emails are spam and which are important. We can assume that each email is represented by n features (e.g., return address, keywords, size, etc.). Then we are given a sample S of emails which have already been labeled as spam or not spam, and are asked to provide a rule to use in the future. For example, our input set S might look like the following.

sales	apply	Mr.	bad spelling	known-sender	spam?
Y	N	Y	Y	N	Y
N	N	N	Y	Y	N
N	Y	N	N	N	Y
Y	N	N	N	Y	N
N	N	Y	N	Y	N
Y	N	N	Y	N	Y
N	N	Y	N	N	N
N	Y	N	Y	N	Y

Given this data, a reasonable hypothesis might be "predict spam if unknown-sender AND (apply OR sales)".

In general, when we are given this kind of question there are two big questions, which are related but distinct. First, how can we automatically generate good hypotheses from the data? It might be a nontrivial computational problem even to find a hypothesis that works on S! Second, how confident are we that our hypothesis will do well in the future? This is some kind of confidence bound or sample complexity bound – for a given learning algorithm, how much data do we need to see before we can make a guarantee about the future?

There's a lot to say about both of these questions, and if you're interested in learning more, I recommend taking Professor Arora's class on machine learning theory! Today we'll be focusing on the first question – how can we automatically generate good hypotheses from the data? Given a sample $S = \{(x^1, y^1), \dots (x^m, y^m)\}$, where the examples x^i are from domain X and the labels y^i are produced by some (unknown) target Boolean function $f : X \to \{0,1\}$, i.e. $y^i = f(x^i)$ for each $i \in [m]$, we want to find a hypothesis h such that $\frac{1}{m}\sum_{i\in[m]} \mathbb{1}[h(x^i) \neq y^i] \leq \epsilon$. We call ϵ the *error* of h (with respect to the sample S).

How hard is it to find a hypothesis h with error at most ε? We will show an important result from learning theory, that finding a hypothesis with error at most ε is not fundamentally more difficult than generating a hypothesis that is only marginally better than random guessing!

Let's begin by defining what we want from a learning algorithm.

Definition 25.2.1 (Strong learner) *We call a learning algorithm $\mathcal{L}$ a* strong learner *for a set of functions $\mathcal{F}$ if for every dataset $S = \{(x^1, y^1), \dots (x^m, y^m)\}$, where the labels y^i are produced by a target function $f \in \mathcal{F}$, $\mathcal{L}$ can find a hypothesis h such that*

$$\frac{1}{m}\sum_{i\in[m]} \mathbb{1}[h(x^i) \neq y^i] \leq \epsilon.$$

Ideally we'd also like our learning algorithm to be efficient – but we'll ignore running time for now.

We will reduce the problem of strong learning to the problem of weak learning:

Definition 25.2.2 (Weak learner) *We call a learning algorithm WkL a γ-*weak learner *for a set of functions $\mathcal{F}$ if for every distribution D over a dataset $S = \{(x^1, y^1), \dots (x^m, y^m)\}$, where the labels y^i are produced by a target function $f \in \mathcal{F}$, WkL can find a hypothesis h such that*

$$\Pr_{(x^i,y^i)\sim D}[h(x^i) \neq y^i] \leq \frac{1}{2} - \gamma.$$

We will show that if we have access to a *weak learner*, then we can construct a *strong learner*. Our strong learner will use the multiplicative weights algorithm from last time to iteratively reweight its sample, and run the weak learner on this reweighted sample. After T iterations, it will aggregate all hypotheses output by the weak learner using a majority vote and use this as its final hypothesis.

Algorithm 1 MW Boosting

```
Let $S = \{(x^1, y^1), \dots (x^m, y^m)\}$ be a dataset.
For $i \in [m]$, let $w_i^1 = 1$ be the weight of $(x_i, y_i)$ at time 1.
for $t = 1, \dots, T$ do
    Let $W^t = \sum_{i \in [m]} w_i^t$ be the total weight at time $t$.
    Let $D^t$ be the distribution over $S$ at time $t$ defined by $D^t(i) = \frac{w_i^t}{W^t}$ for all $i \in [m]$.
    Run $WkL$ on sample $S$ with distribution $D^t$ and let $h_t$ be the hypothesis output by $WkL$
    For all $j \in [m]$, let the loss of $h_t$ on example $j$ be defined $\ell_j^t = \begin{cases} 0 & h_t(x_j) \neq y_j \\ 1 & \text{otherwise} \end{cases}$
    Update weights: $w_j^{t+1} \leftarrow w_j^t \cdot e^{-\gamma \ell_j^t}$ for all $j \in [m]$
end for
return $h(x) =$ majority vote of $\{h_1(x), \dots, h_T(x)\}$
```

Before we analyze convergence of this algorithm, let's first compare to the multiplicative weights algorithm applied to online learning. Last time, the algorithm was using this exponential update rule to reweight actions based on their loss. Actions with large loss are given smaller weight, and actions with small loss are given larger weight. In this boosting algorithm, we are using the same update rule, but instead of reweighting actions, we are reweighting examples based on how many hypotheses have correctly classified these examples so far. Maybe counterintuitively, examples on which many hypotheses are correct will suffer large loss, and so be downweighted. Examples on which many hypotheses are incorrect will suffer small loss, and so be upweighted. Why do we want to upweight "bad" examples in boosting? Well our weak learner is guaranteed to give us a hypothesis that does better than random guessing on the reweighted distribution, so we want to give more weight to examples on which we've made many mistakes so far, to ensure that in expectation we make progress towards correctly classifying these bad examples.

Theorem 25.2.3 *Let $\varepsilon \in (0, 1)$ and $\gamma \in (0, \frac{1}{2})$. Let $\mathcal{F}$ be a class of Boolean functions and let $S = \{(x^1, y^1), \dots (x^m, y^m)\}$ be a dataset for which $y^i = f(x^i)$ for some $f \in \mathcal{F}$. Given access to a γ-weak learner WkL for $\mathcal{F}$ and run for $T \geq \frac{2\log(1/\varepsilon)}{\gamma^2}$ rounds, the MW Boosting algorithm will output a hypothesis h with error at most ε.*

Proof: The proof will look fairly similar to the proof of the multiplicative weights algorithm applied to online learning. We will use the total weight W^{T+1} on all examples in S at the final round to upper bound the number of examples h misclassifies. We will also use the guarantee from our weak learner to show that the total weight W^t must decrease every round, and therefore after $T \in O(\frac{\log(1/\varepsilon)}{\gamma^2})$ rounds, we must have a hypothesis that does not make too many mistakes.

We'll begin by lower-bounding W^{T+1} in terms of the number of examples h misclassifies. Let $B = \{i \in [m] : h(x^i) \neq y^i\}$ be the set of examples that h misclassifies. Since the final hypothesis is a majority vote of the hypotheses output by the weak learner, it only misclassifies an example if at least $T/2$ of the hypotheses output by the weak learner misclassify that example. This means that for every $i \in B$,at most $T/2$ of the losses $\ell_i^t = 1$ for a misclassified example, and the remainder must be $\ell_i^t = 0$. So the weight on this example must be at least $w_i^{T+1} \geq w_i^1 e^{-\gamma T/2} = e^{-\gamma T/2}$. Summing over all $i \in B$, we have

$$W^{T+1} \geq |B| e^{-\gamma T/2}.$$

Now we want to upper-bound W^{T+1} in terms of T. The approach here will be similar to the upper bound we proved for the multiplicative weights algorithm applied to online learning, in that we will obtain an upper-bound on W^{t+1} in terms of W^t and the expected loss at round t. Then, by the guarantee of our weak learner, we know that our expected loss at each iteration must be at least $\frac{1}{2} + \gamma$, and so the total weight must decrease round by round.

$$\begin{aligned}
W_{t+1} &= \sum_{i \in [m]} w_i^{t+1} && \text{by definition of } W_{t+1} \\
&= \sum_{i \in [m]} w_i^t \cdot e^{-\gamma \ell_i^t} && \text{by definition of } w_i^{t+1} \\
&= W_t \sum_{i \in [m]} \tfrac{w_i^t}{W_t} e^{-\gamma \ell_i^t} && \text{multiply by } \frac{W_t}{W_t} \\
&= W_t \mathop{\mathbb{E}}_{i \sim D^t} [e^{-\gamma \ell_i^t}] && \text{by definition of } D^t \\
&= W_t (\mathop{\mathbb{E}}_{i \sim D^t} [e^{-\gamma \ell_i^t} \mid \ell_i^t = 0] \cdot (1 - \Pr_{i \sim D^t}[\ell_i^t = 1]) \\
&\qquad + \mathop{\mathbb{E}}_{i \sim D^t} [e^{-\gamma \ell_i^t} \mid \ell_i^t = 1] \cdot \Pr_{i \sim D^t}[\ell_i^t = 1]) && \text{law of total expectation} \\
&= W_t (1 - \Pr_{i \sim D^t}[\ell_i^t = 1] + e^{-\gamma} \Pr_{i \sim D^t}[\ell_i^t = 1])) \\
&\leq W_t \exp(- \Pr_{i \sim D^t}[\ell_i^t = 1] + e^{-\gamma} \Pr_{i \sim D^t}[\ell_i^t = 1])) && 1 - x \leq e^{-x} \\
&= W_t \exp(- \Pr_{i \sim D^t}[\ell_i^t = 1](1 - e^{-\gamma})) && \text{factor out probability} \\
&\leq W_t \exp(-(\tfrac{1}{2} + \gamma)(1 - e^{-\gamma})) && \text{from the weak learner guarantee} \\
&\leq W_t \exp(-(\tfrac{1}{2} + \gamma)(\gamma - \gamma^2/2)) && \text{using } e^{-\gamma} \leq 1 - \gamma + \gamma^2/2 \\
&= W_t \exp(-\tfrac{\gamma}{2} - \tfrac{3\gamma^2}{4} + \tfrac{\gamma^3}{2})
\end{aligned}$$

This means that after T iterations, the total weight W^{T+1} is at most

$$W^{T+1} \leq W_1 \exp(-\tfrac{\gamma T}{2} - \tfrac{3\gamma^2 T}{4} + \tfrac{\gamma^3 T}{2}) = m \exp(-\tfrac{\gamma T}{2} - \tfrac{3\gamma^2 T}{4} + \tfrac{\gamma^3 T}{2}).$$

Putting our upper and lower bound together, we get

$$\begin{aligned}
|B| \exp(-\gamma T/2) &\leq W^{T+1} \leq m \exp(-\tfrac{\gamma T}{2} - \tfrac{3\gamma^2 T}{4} + \tfrac{\gamma^3 T}{2}) \\
\Rightarrow \frac{|B|}{m} &\leq \exp(-\tfrac{3\gamma^2 T}{4} + \tfrac{\gamma^3 T}{2}) \\
&= \exp(-\frac{\gamma^2 T}{2}(\frac{3}{2} - \gamma)) && \text{factor out } \gamma^2 T/2 \\
&\leq \exp(-\frac{\gamma^2 T}{2}) && \gamma \leq 1/2
\end{aligned}$$

Therefore to ensure $\frac{|B|}{m} \leq \varepsilon$, it suffices to take $\gamma^2 T/2 \geq \log(1/\varepsilon)$, or $T \geq \frac{2\log(1/\varepsilon)}{\gamma^2}$. ■

Boosting algorithms are a powerful tool in machine learning, but they're just one additional application of the multiplicative weights algorithm (and more generally, no-regret algorithms). We sadly don't have time to discuss additional applications in much depth, but multiplicative weights has uses in a variety of contexts including:

- approximately solving linear programs
- game theory (computing approximate Nash equilibria in 2-player zero-sum games)
- hardness amplification (building strong cryptographic primitives from weak ones)
- network congestion control
- computational geometry