Notes on Weighted Majority (COL 758)

1 The weighted majority algorithm

We describe the setting for the "on-line learning from experts" problem. We are given N experts, $e_1, \ldots, e_n$. At each time t, the expert e_i incurs loss $y_i^{(t)}$. We assume that the loss lies in the range $[-1, 1]$. At each time t, the algorithm needs to pick one of the experts (without knowing the vector $y^{(t)}$). The algorithm maintains weights $w_i^{(t)}$ of the experts. At time t, it picks an expert e_i with probability proportional to its weight $w_i^{(t)}$. After it picks an expert, it gets to know the loss vector $y^{(t)}$. It updates the weights of the experts for the next time step, and so on. The online learning algorithm is described in Figure 1. Here, η is a parameter between 0 and 1/2. Note that the algorithm decreases the weights of experts who incur loss and increases the weights of experts who incur gain (or negative loss). Note that T denotes the time horizon over which we run the algorithm.

```
Algorithm Learn From Experts:
Initialize $w_i^{(1)} \leftarrow 1$ for all experts $e_i$.
For $t = 1, 2, \ldots, T$
    Pick expert $e_i$ with probability proportional to $w_i^{(t)}$.
    Update $w_i^{(t+1)} \leftarrow (1 - \eta y_i^{(t)}) \cdot w_i^{(t)}$ for all experts $e_i$.
```

Figure 1: On-line algorithm for learning from experts.

Theorem 1 *The total expected loss of the algorithm is at most*

$$\sum_{t=1}^{T} y_i^{(t)} + \eta \cdot \sum_{t=1}^{T} |y_i^{(t)}| + \frac{\log n}{\eta},$$

where i is any index between 1 and n.

Note that the expression $\sum_{t=1}^{T} y_i^{(t)}$ denotes the total loss of expert e_i. Thus, the theorem says that the expected loss of the algorithm is at most the loss incurred by the best expert plus a small additive term.
Proof: The idea behind the proof is to look at the potential function

$$\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)},$$

which is the total weight of all the experts at time t. Whenever our algorithm incurs high loss, it must be the case that most of the weight lies with experts who incur high loss at time t. Therefore, the total weight of the experts should decrease by a good amount. We can formalize this reasoning as follows :

$$\Phi^{(t+1)} = \sum_{i=1}^{n} w_i^{(t+1)} = \sum_{i=1}^{n} w_i^{(t)}(1 - \eta y_i^{(t)}) = \Phi^{(t)} \cdot \left(1 - \eta \sum_{i=1}^{n} \frac{y_i^{(t)} w_i^{(t)}}{\Phi^{(t)}}\right).$$

Let $l^{(t)}$ denote the expected loss incurred by the algorithm at time t. Observe that $l^{(t)}$ is exactly $\sum_{i=1}^{n} \frac{y_i^{(t)} w_i^{(t)}}{\Phi^{(t)}}$. Therefore, we have

$$\Phi^{(t+1)} = \Phi^{(t)} \cdot (1 - \eta l^{(t)}) \leq \Phi^{(t)} e^{-\eta l^{(t)}}.$$

Thus, we see that if $l^{(t)}$ is large, the potential drops by a large amount. The above inequality also shows that

$$\Phi^{(T+1)} = \Phi^{(1)} \cdot \prod_{i=1}^{T} e^{-\eta l^{(t)}} = n \cdot e^{-\eta \sum_{t=1}^{T} l^{(t)}}. \tag{1}$$

At the same time, we know that for any fixed expert i, $\Phi^{(T+1)} \geq w_i^{(T+1)}$. If the algorithm incurs large loss, the potential $\Phi^{(T+1)}$ will be very small, which implies that the weight of e_i is very small. Therefore, e_i must have incurred large loss as well. For the expert e_i, let I^+ denote the time time steps t for which $y_i^{(t)} \geq 0$, and T^- be the remaining time steps (i.e., when $y_i^{(t)} < 0$). Observe that

$$w_i^{(T+1)} = \prod_{t=1}^{T} (1 - \eta y_i^{(t)}) = \prod_{t \in I^+} (1 - \eta y_i^{(t)}) \cdot \prod_{t \in I^-} (1 - \eta y_i^{(t)}) \leq (1-\eta)^{l_i^+} \cdot (1+\eta)^{-l_i^-},$$

where l_i^+ denotes $\sum_{i \in I^+} y_i^{(t)}$ and l_i^- denotes $\sum_{i \in I^-} y_i^{(t)}$, and we used the following easy to check facts:

$$(1 - \eta x) \leq (1-\eta)^x, (1 + \eta x) \leq (1+\eta)^x, \text{for } 0 < x < 1.$$

Since $\Phi^{(T+1)} \geq w_i^{(T+1)}$, combining this fact with (1), we get

$$\sum_{t=1}^{T} l^{(t)} \leq \frac{\log n}{\eta} - \frac{\log(1-\eta)}{\eta} l_i^+ + \frac{\log(1+\eta)}{\eta} l_i^- \leq \frac{\log n}{\eta} + (l_i^+ + l_i^-) + \eta(l_i^+ - l_i^-),$$

where we have used the following facts (which are true because $0 \leq \eta \leq 1/2$) :

$$-\log(1-\eta) \leq \eta + \eta^2, \ \log(1+\eta) \leq \eta - \eta^2.$$

Finally observe that $l_i^+ + l_i^-$ is exactly the total loss incurred by e_i, i.e., $\sum_{t=1}^{T} y_i^{(t)}$, and $l_i^+ - l_i^-$ is $\sum_{t=1}^{T} |y_i^{(t)}|$. ■

We get the following corollary.

Corollary 1 *With a suitable choice of η, the total expected loss of the algorithm is at most*

$$\sum_{t=1}^{T} y_i^{(t)} + \sqrt{T \log n},$$

where i is any index between 1 and n.

Proof: We pick $\eta = \sqrt{T/\log n}$, and observe that $\sum_{t=1}^{T} |y_i^{(t)}| \leq T$. ■

2 Applications to Solving Linear Programs

We show how to use the above framework to solve a special class of linear program, called fractional set cover. The framework is more powerful, and can be extended to more general LPs. Here we are given n variables $x_1, \ldots, x_m$, a cost vector $(c_1, \ldots, c_m)$, and subsets $S_1, \ldots, S_n$ of $U = \{1, \ldots, m\}$. The LP can be written as

$$\begin{array}{rcll} \min. & & \sum_{j=1}^{m} c_j x_j & \\ & \text{s.t.} & & \\ \sum_{j \in S_i} x_j & \geq & 1 & (\text{ constraint } E_i) \\ x_j & \geq & 0 & \text{for all } j \end{array}$$

We will use E_i to denote the constraint for set S_i, where E_i should be thought of as the i^{th} expert. Instead of solving a minimisation problem, we will solve a *feasibility* problem – given a number T, we will find a solution $x_1, \ldots, x_n$ which satisfies the following conditions (assuming such a solution exists):

$$\begin{array}{rcll} \sum_{j=1}^{m} c_j x_j & \leq & T & \\ \sum_{j \in S_i} x_j & \geq & 1 & (\text{ constraint } E_i) \\ x_j & \geq & 0 & \text{for all } j \end{array}$$

We can solve the minimization problem by trying different values of T. The minimum possible value of T is 0, and the maximum possible value is $mc_{\max}$. Therefore, we can solve the optimization problem by trying $\log(mc_{\max})$ different values of T. We focus on the feasibility problem now. Another way of stating this problem is as follows : let $\mathcal{P}$ denote the set of vectors $(x_1, \ldots, x_n)$ satisfying the following constraints:

$$c_1 x_1 + \ldots + c_m x_m \leq T, \; x_1, \ldots, x_m \geq 0.$$

The feasibility problem asks us to find a vector in $\mathcal{P}$ which satisfies each of the constraints $E_1, \ldots, E_n$. Note that if we had just one constraint, say E_1, then the problem is easy – just pick the cheapest variable in S_1 to an extent of 1 unit. We will reduce the problem to learning from experts. Our algorithm will run through several iterations. In each iteration, it will find a candidate solution (which may not satisfy all the constraints), and finally output the average of all these solutions. The idea behind the candidate solution in an iteration is to pick one expert (or constraint) and just solve the feasibility problem with respect to this constraint only. Which expert should we pick? We would like this to be the constraint which is getting violated a lot by the current solution, so that in the overall average, we more or less satisfy all the constraints. For this plan to work, we should assign weights to constraints in such a manner that the more violated constraints get *higher* weights – recall that the learning by experts algorithms has higher probability of picking a higher weight expert. Therefore, given a solution x, the loss corresponding to E_i is proportional to $(\sum_{j \in S_i} x_j - 1)$ – higher violation implies less (in fact, negative) loss. Since we are required to keep the loss in the range $[-1, 1]$, we will define the loss of E_i (corresponding to a solution x) as $\frac{1 - \sum_{j \in S_i} x_j}{m}$. One final comment: the learning from experts algorithm does not deterministically pick a single expert, but maintains a distribution over the experts. It turns out that the corresponding choice here would to pick a weighted combination of constraints (rather than a single constraint). Given the constraints $E_1, \ldots, E_n$, and non-negative weights $w_1, \ldots, w_n$, define $w_1 E_1 + \ldots + w_n E_n$ to be the constraint obtained by multiplying E_i by w_i on both sides and then adding up all these constraints. Clearly, if there is a solution $x \in \mathcal{P}$ which satisfies all the constraints, then there would be a solution $x \in \mathcal{P}$ which satisfies any (positive) linear combination of these constraints. But the latter is a much simpler problem – any such constraint will look like $a_1 x_1 + \ldots + a_n x_n \geq b$, where $a_1, \ldots, a_n, b \geq 0$. It is easy to see that the minimum possible value of $c_1 x_1 + \ldots + c_m x_m$ subject to this constraint is $c_i \cdot \frac{b}{a_i}$, where i denotes the index for which the ratio c_i/a_i is minimum. Thus, we just need to check if $c_i \cdot \frac{b}{a_i} \leq T$ – if so, then this is a feasible solution, else there is no solution.

The algorithm is described in Figure 2. We now analyze this algorithm. We first compute the expected

```
Algorithm Solve LP:
1. Initialize $w_i^{(1)} \leftarrow 1$ for all constraints $E_i$.
2. For $t = 1, 2, \ldots, T$
    (i) Let $x^{(t)}$ be a solution in $\mathcal{P}$ which satisfies the constraint $\sum_{i=1}^n w_i^{(t)} E_i$
            If there is no such solution, STOP and declare that the LP is infeasible.
    (ii) Update $w_i^{(t+1)} \leftarrow (1 - \eta y_i^{(t)}) \cdot w_i^{(t)}$ for all experts $E_i$, where $y_i^{(t)} = \frac{\sum_{j \in S_i} x_j - 1}{m}$.
3. Output $\bar{x} \leftarrow 1/T \cdot \sum_{t=1}^T x^{(t)}$.
```

Figure 2: On-line algorithm for solving LP.

loss of the algorithm at time t. As before, let $\Phi^{(t)}$ denote $\sum_{i=1}^n w_i^{(t)}$. Observe that the expected loss of the algorithm at time t is is equal to

$$\sum_{i=1}^n \frac{w_i^{(t)} y_i^{(t)}}{\Phi^{(t)}} = \sum_{i=1}^n \frac{w_i^{(t)} (\sum_{j \in S_i} x_j^{(t)} - 1)}{m \Phi^{(t)}} \geq 0,$$

where the last inequality follows from the fact that the numerator corresponds to the constraint $w_1 E_1 + \ldots + w_n E_n$, and $x^{(t)}$ satisfies this constraint. Therefore, the total expected loss of the algorithm is non-negative. Corollary 1 now shows that the total loss of any constraint E_i is at least $-\sqrt{T \log n}$. But the total loss of E_i can be expressed as

$$\sum_{t=1}^T y_i^{(t)} = \sum_{t=1}^T \frac{1 - \sum_{j \in S_i} x_j}{m} = \frac{T}{m} \cdot (\sum_{j \in S_i} \bar{x}_j - 1).$$

Thus, we see that

$$\sum_{j \in S_i} \bar{x}_j - 1 \geq -m \sqrt{\frac{\log n}{T}} \geq -\delta,$$

if we pick T to be $m^2/\delta^2 \cdot \log n$. Rearranging terms, we see that $\bar{x}/(1-\delta)$ satisfies all the constraints, and the total cost of this solution is at most $T/(1-\delta)$. It follows that this algorithm will find a solution of cost at most $T^*/(1-\delta)$, where $T^\star$ is the minimum cost of any solution to the LP.

How much time did the algorithm take? Each of the iterations takes $O(mn)$ time. Thus the total running time for a fixed value of T is $O(m^3 n/\delta^2 \cdot \log n)$. Finally, we will try $\log(mc_{\max})$ different values of T. Therefore, the total running time is $O(m^3 n/\delta^2 \cdot \log n \cdot \log(mc_{\max}))$.

Exercise 1 *Suppose that we pick a constraint E_i with probability proportional to $w_i^{(t)}$ in Step 2(i) of the above algorithm (instead of picking a weighted linear combination of the experts). Will the analysis still work ?*

Exercise 2 *Use Theorem 1 instead of Corollary 1 to show that it suffices to pick $T = O(m/\delta^2 \cdot \log n)$ in the analysis above.*

Exercise 3 *(taken from notes by Shuchi Chawla) Here is a variation on the deterministic Weighted-Majority algorithm, designed to make it more adaptive.*

(a) Each expert begins with weight 1 (as before).

(b) At every step, we predict the result of a weighted-majority vote of the experts (as before).

(c) At every step, for every expert that makes a mistake, we penalize it by dividing its weight by 2, but only if its weight was at least 1/4 of the average weight of experts.

Prove that in any contiguous block of steps (e.g., the 51st step through the 77th step), the number of mistakes made by the algorithm is at most $O(m + \log n)$, where m is the number of mistakes made by the best expert in that block, and n is the total number of experts.