Prediction with Expert Advice

Set up

- N **experts**
- T **time periods, in each time period** $t \leq T$ **,**
  - **Each expert** i **makes a** $0, 1$ **prediction** $\mathcal{E}_i^t$ **.**
  - **We, after viewing the vector** $\mathcal{E}^t$ **, make a prediction** p^t **.**
  - **We learn the true outcome** o^t **.**
- **The objective is to minimize the number of mistakes,** $\Sigma_{t=1}^{T} |p^t - o^t|$

How do we evaluate how well we did?

Prediction with Expert Advice

Set up

- N **experts**
- T **time periods, in each time period** $t \leq T$ **,**
  - **Each expert** i **makes a** $0, 1$ **prediction** $\mathcal{E}_i^t$ **.**
  - **We, after viewing the vector** $\mathcal{E}^t$ **, make a prediction** p^t **.**
  - **We learn the true outcome** o^t **.**
- **The objective is to minimize the number of mistakes,** $M = \Sigma_{t=1}^{T} |p^t - o^t|$

How do evaluate how well we did?

- **We will assume that the experts predictions are chosen adversarily**
- **We will compare ourselve to the best expert, in retrospect (regret),** $\min_i \Sigma_t |o^t - \mathcal{E}_i^t|$

First evaluations

Perfect expert: If there is a perfect expert, then there is an algorithm which makes at most $\log_2 N$ mistakes.

Proof idea: The algorithm is to take the majority vote of the experts who have never made an error yet.

Imperfect expert: If the best expert makes m mistakes, then there is an algorithm making at most

$$m(\log_2 N + 1) + \log_2 N$$

mistakes.

Proof idea: Divide time into epochs and use the previous algorithm, restarting an epoch as soon as every expert has made at least one error during that epoch.

Can we do better?

Weighted Majority Algorithm

1. **Assign a weight w_i to each expert i. Initialize $w_i = 1$.**
2. **At time t,**
   - **Predict according to the *weighted majority* (compare the weights of the 0 experts vs. the 1 experts)**
   - **For each expert i who made a mistake, $w_i^{t+1} = (1/2)w_i^t$**

Theorem: **If m_i is the number of mistakes made by expert i before time T, and M is the number of mistakes made by the algorithm, then**

$$M \leq 2.41(m_i + \log_2 N).$$

Weighted Majority Algorithm

1. Assign a weight w_i to each expert i. Initialize $w_i = 1$.
2. At time t,
   - Predict according to the *weighted majority* (compare the weights of the 0 experts vs. the 1 experts)
   - For each expert i who made a mistake, $w_i^{t+1} = (1/2)w_i^t$

Theorem: If m_i is the number of mistakes made by expert i before time T, and M is the number of mistakes made by the algorithm, then

$$M \leq 2.41(m_i + \log_2 N).$$

Corollary: If we multiply weights by $1 - \epsilon$ instead of $1/2$ when there is a mistake, we get a bound of

$$M \leq 2(1+\epsilon)m_i + \frac{O(\log_2 N)}{\epsilon}.$$

Remark: Can't go below 2 with deterministic algorithms, but can consider randomized ones.

Randomized Multiplicative Weights

Recall $\Phi^t = \Sigma_i \, w_i^t$.

Define

$$P0 = \frac{\Sigma_{i:\mathcal{E}_i^t=0} \, w_i^t}{\Phi_t} \qquad P1 = \frac{\Sigma_{i:\mathcal{E}_i^t=1} \, w_i^t}{\Phi_t}$$

Algorithm

1. **Assign a weight** w_i **to each expert** i **. Initialize** $w_i = 1$ **.**
2. **At time** t **,**
   - **Predict according to the *randomized weighted majority*. Predict** 0 **with probability** $P0$ **and** 1 **with probability** $P1$ **.**
   - **For each expert** i **who made a mistake,** $w_i^{t+1} = (1-\epsilon)w_i^t$

Theorem: **Given any** $\epsilon > 0$ **, if** m_i **is the number of mistakes made by expert** i **before time** T **, and** q **is the number of mistakes made by the algorithm, then**

$$E[M] \leq (1+\epsilon)m_i + \frac{O(\log_2 N)}{\epsilon}$$

Remarks

- The regret is the difference between the algorith and the best expert in retrospect, and is $\epsilon m_i + \frac{O(\log_2 N)}{\epsilon}$.
- Need to be careful about the adversary with a randomized algorithm:
  - An oblivious adversary has to choose the entire sequence $\mathcal{E}^1, o^1, \mathcal{E}^2, o^2, \ldots$ in advance.
  - An adpative adversary can choose $\mathcal{E}^t$ and o^t , knowing $\mathcal{E}^1, o^1, \mathcal{E}^2, o^2, \ldots, ce^{t-1}, o^{t-1}$.

A more general situation

- **At each step, we choose a "probability vector"** $p^t = (p_1^t, \ldots, p_N^t)$ **, with** $p_i^t \geq 0,\ \Sigma_i p_i^t = 1$ **.**
- **At each step the adversary produced a loss vector,** $l^t = (l_1^t, \ldots, l_N^t) \in [-1, 1]^N$ **.**
- **Our cost (loss) at step** t **is** $L^t = \langle p^t, l^t \rangle$ **, where we use** $\langle \cdot, \cdot \rangle$ **to denote the inner product of 2 vectors.**
- **We compare our algorithm against the best static strategy.**
- **Note that our algorithm is not necessarily randomized.**

Algorithm Hedge:

- **Maintain Multiplicative Weights updated with rule** $w_i^{t+1} = w_i^t e^{-\epsilon l_i^t}$ **.**
- **Let** $\Phi^t = \Sigma_i w_i^t$ **.**
- **Set probabilities,** $p_i^t = w_i^t / \Phi^t$ **.**

Theorem: **Let** $\epsilon \leq 1$ **. For all times** T **, all sequences of loss vectors** $l^1, \ldots, l^T$ **, and all** i **, algorithm Hedge is a deterministic algorithm s.t.**

$$\sum_{t=1}^{T} L^t \leq \sum_{t=1}^{T} l_i^t + \epsilon T + \ln N/\epsilon$$

Corollary

Theorem: Let $\epsilon \leq 1$. For all times T, all sequences of loss vectors $l^1, \ldots, l^T$, and all i, algorithm Hedge is a deterministic algorithm produceing a probability vector s.t.

$$\sum_{t=1}^{T} L^t \leq \sum_{t=1}^{T} l_i^t + \epsilon T + \ln N/\epsilon$$

Corollary A: If $T \geq 4 \ln N/\epsilon^2$, then

$$\frac{\Sigma_{t=1}^{T} L^t}{T} \leq \frac{\Sigma_{t=1}^{T} l_i^t}{T} + \epsilon$$

- We can scale our loss vectors to be in $[-\rho, \rho]^N$.

Corollary B:

Let $\epsilon \leq 1$. For all times $T \geq 4 \ln N \rho^2/\epsilon^2$, all sequences of loss vectors $l^1, \ldots, l^T \in [-\rho, \rho]^N$, and all i, algorithm Hedge is a deterministic algorithm produceing a probability vector s.t.

$$\frac{\Sigma_{t=1}^{T} L^t}{T} \leq \frac{\Sigma_{t=1}^{T} l_i^t}{T} + \epsilon$$

Approximately Solving Packing LPs

Problem:

$$\min\langle c, x\rangle$$

$$Ax \geq b \quad x \geq 0$$

- **In a packing LP,** A, b, c **are all non-negative.**
- **We will assume that we know OPT, the optimal solution to the LP. We can use binary search to know OPT up to some accuracy.**
- **Let** $K = \{x \in R^n : x \geq 0, \langle c, x\rangle = OPT\}$ **be the set of optimal solutions.**

Restatement of approximate solution to LP:
Find an $x \in K$ **s.t.** $\langle a_i, x\rangle \geq b_i - \epsilon \quad \forall i = 1, \ldots, m$

Oracles

Warmup Lemma: Given a single constraint $\langle \alpha, x \rangle \geq \beta$, there exists a fast oracle to find $x \in K$ s.t. $\langle \alpha, x \rangle \geq \beta$ or to say that none exists.

Oracle: We will assume that we have an Oracle that allows us to optimize quickly over the easy constraints, find a solution in K that satsifies some easy constraints. As we have seen, we can optimize over one constraint. We can also optimize over "nice" constraints, such as shortest path constraints..

Strategy for Approximately Solving LP

- Use Hedge to update weights on constraints.
- Use the Oracle to produce a solution to a convex combination of the constraints
- Update weights depending on how badly constraints are violated.
- Repeat.

Width: $\rho = \max_{x \in K, i} \{ |\langle a_i, x \rangle - b_i \}$ is the maximum possible violation of a constraint by a feasible solution and will affect the running time.

Algorithm A

1. $p^1 = (1/m, \ldots, 1/m)$

2. For $t = 1$ **to** $\rho^2 \ln m/\epsilon^2$

(a) Use the ORACLE to either find an x^t **s.t.** $\Sigma_{i=1}^m \langle p_i^t a_i, x^t \rangle \geq \Sigma_{i=1}^m p_i^t b_i$ **or return infeasible**

(b) $l_i^t = \langle a_i, x^t \rangle - b_i$

(c) Call $HEDGE(\epsilon)$ **to update** p^{t+1}

3. Return $\hat{x} = (x_1 + \cdots + x_T)/T$

- **If the algorithm ever returns infeasible, then the original problems is infeasible (because feasible problems always can satsify a convex combination of the constraints).**
- **The ORACLE is just a one constraint LP.**
- **If you think about the solution as** T **grows, you see that it is gradually approaching a feasible solution, by putting more weights on unsatisfied constraints.**

Analysis

Outline **Consider Corollary B that applies to Hedge. After** $T = \rho^2 \ln m/\epsilon^2$ **rounds of Hedge,**

$$\frac{\Sigma_{t=1}^{T} \langle p^t, l^t \rangle}{T} \leq \frac{\Sigma_{t=1}^{T} l_i^t}{T} + \epsilon$$

.

- **We will show that the left hand side is non-negative.**
- **We will show that the right hand side is bound by** $a_i \hat{x} - b_i + \epsilon$ **.**
- **We will conclude that** $a_i \hat{x} \geq b_i - \epsilon$ **which implies approximate feasibliity.**

Multicommodity Flow

Consider the maximization version of multicommodity flow. You can express it (inefficiently) as being given a set of paths P, and you have to put flow on the paths f_p, subject to capacity constraints on the edges.

$$\begin{aligned} \max \sum_{p \in P} f_p & \\ \sum_{p \ni e} f_p & \leq u_e \quad \forall e \in E \\ f_p & \geq 0 \quad \forall p \in P \end{aligned}$$

- **Let F^* be the amount of flow in the optimal solution. We now have the problem**

$$\exists p \in \mathcal{P} : \forall e \in E : \sum_{p \ni e} f_p \leq u_e,$$

where

$$\mathcal{P} = \left\{ f : \forall p \in P : f_p \geq 0, \sum_{p \in P} f_p = F^* \right\}.$$

- **What does multiplicative weights look like for this packing problem?**

Multiplicative Weights for Multicommodity Flow

- Maintain edge weights w_e^t
- Find flow which minimizes:

$$\sum_e w_e^t \sum_{p \ni e} f_p/u_e = \sum_p f_p \sum_{e \in p} w_e^t/u_e$$

- This is a shortest path problem where edge e has length w_e^t/u_e .
- Each iteration picks a path which is shortest, and puts F^* flow on that path.
- Final flow is an average of all the paths and hence is a flow of value F^* .
- Width is not polynomial.

$$\rho = \max_{f \in \mathcal{P}} \max_e \sum_{p \ni e} f_p/u_e = F^*/u_{\min}$$

- Each iteration is a shortest path but algorithm is not polynomial

Ideas for a Polynomial Algorithm

- **Reduce width by only considering feasible flows and not flows of value F^* , which immediately reduces width.**
- **Send an amount of flow on path p determined by minimum capacity edge on path p .**
- **Update weights only on path with flow sent on it.**
- **Stop when an edge gets too much congestion.**
- **Running time is $O(\epsilon^{-2} m \log n T_{sp})$ where T_{sp} is the time for a shortest path computation.**