18.409 An Algorithmist's Toolkit December 10, 2009

Lecture 25

Lecturer: Jonathan Kelner *Scribe: Nikolaos Trichakis*

Multiplicative Weights

In this lecture, we will study various applications of the theory of Multiplicative Weights (MW). In this section, we briefly review the general version of the MW algorithm that we studied in the previous lecture. The following sections then show how the theory can be applied to approximately solve zero-sum games and linear programs, and how it connects with the theory of boosting and approximation algorithms.

We have n experts who predict the outcome of an event in consecutive rounds. Suppose that in each round there are P different outcomes for the event. If outcome j realizes, expert i pays a penalty of $M(i,j)$. An important paremeter will prove to be the maximum allowable magnitude for the penalty. For that, let $M(i,j) \in [-\ell, \rho]$, with $0 \leq \ell \leq \rho$, where ρ is the *width*. Our goal is to devise a strategy that dictates which expert's reccomendation to follow, in order to achieve an expected avegare penalty that is not much worse than that of the best expert (in hindsight).

The strategy that we analyzed in the previous lecture is as follows. We maintain for each expert a scalar weight, which can be thought of as a quality score. Then, at each round we choose to follow the recommendation of a specific expert with probability that is proportional to her weight. After the outcome is realized, we update the weights of each expert accordingly. In mathematical terms, let w_i^t be the weight of the ith epxert at the beginning of round t. Then, the MW algorithm is

0. Initialize $w_i^1 = 1$, for all i.
1. At step t,
   a. Follow the recommedation of the ith epxert with probability p_i^t, where
   $$p_i^t = \frac{w_i^t}{\sum_j w_j^t}.$$
   b. Let $j^t \in P$ denote the outcome of the event at round t, and $D^t = \{p_1^t, \ldots, p_n^t\}$ the distribution we used above to select an expert. Our penalty is denoted by $M(D^t, j^t)$, and is equal to $M(i, j^t)$, where i is the selected expert.
   c. Update the weights as follows:
   $$w_i^{t+1} = \begin{cases} w_i^t(1-\epsilon)^{M(i,j^t)/\rho} & \text{if } M(i,j^t) \geq 0 \\ w_i^t(1+\epsilon)^{-M(i,j^t)/\rho} & \text{if } M(i,j^t) < 0. \end{cases}$$

In the previous lecture we argued that for any $\delta > 0$, for $\epsilon \leq \min\left\{\frac{1}{2}, \frac{\delta}{4\rho}\right\}$ and after $T = \frac{16\rho^2 \log(n)}{\delta^2}$ rounds and for all i, the average penalty we get per round obeys:

$$\frac{\sum_{t=1}^T M(D^t, j^t)}{T} \leq \delta + \frac{\sum_{t=1}^T M(i, j^t)}{T}.$$

In particular our average penalty per round is at most δ bigger than the average penalty of the best expert.

Zero-Sum Games

There are two players, labeled as the "row" player R and the "column" player C. Each player has a finite set of actions that he can follow. At each round, player R pays player C an amount that depends on the actions of the two players. In particular, if R plays action i and C plays j, the payoff from R to C is $M(i,j)$. We assume that the payoffs are normalized, such that $M(i,j) \in [0,1]$. Naturally, player R tries to minimize the payoff, whereas player C tries to maximize it.

Each player can follow a pure strategy, which dictates a single action to be played repeatedly, or a mixed strategy, under which the player has a fixed probability distribution over actions, and chooses actions randomly according to it. One might expect that the order in which players choose their actions might play a role, since knowledge of your opponent's strategy helps you to adopt your strategy appropriately. If we let D and P to be the row and column mixed strategies respectively, the von Neumann's minimax Theorem says that in this game, the order of choosing actions is actually indifferent for the players. Mathematically,

$$\lambda^\star := \min_D \max_j M(D,j) = \max_P \min_i M(i,P),$$

where $\lambda^\star$ is the so called *value* of the game. Our goal is to approximate this value, up to some additive error δ.

We deploy the MW algorithm as follows. Let pure strategies for R correspond to experts, and pure strategies for C correspond to events. Then, the penalty paid by expert i in case of event j is exactly the payoff from R to C, if they follow strategies i and j accordingly, that is $M(i,j)$. Assume also that for a mixed strategy D, we can efficiently compute the column strategy j that maximizes $M(D,j)$ (a quantity eventually $\geq \lambda^\star$). At step t of the MW algorithm, we choose a distribution D^t over experts, which then corresponds to a mixed strategy for R. Given D^t, we compute the worst possible event, which is the column strategy j^t that maximizes $M(D^t, j^t)$.

To see why this approach yields an approximation to $\lambda^\star$, first note that for any distribution D,

$$\sum_t M(D, j^t) \geq \min_i \sum_t M(i, j^t), \tag{1}$$

since a distribution is just a weighted average of pure strategies. Furthermore, as we argued above we have

$$M(D^t, j^t) \geq \lambda^\star, \tag{2}$$

since we pick the payoff-maximizing column strategy. According to the MW theory, after $T = \frac{16\log(n)}{\delta^2}$ rounds and for any distribution D we have

$$\lambda^\star \leq \frac{\sum_{t=1}^T M(D^t, j^t)}{T} \leq \delta + \min_i \left\{ \frac{\sum_{t=1}^T M(i, j^t)}{T} \right\} \leq \delta + \frac{\sum_{t=1}^T M(D, j^t)}{T}.$$

The first inequality follows from (2) and the second from (1). Since the above is true for any distribution D, it is also true for the optimal distribution, and hence

$$\lambda^\star \leq \frac{\sum_{t=1}^T M(D^t, j^t)}{T} \leq \delta + \lambda^\star.$$

This demonstrates that the average penalty of the algorithm is an approximation of the value of the game, within and additive positive term of δ. Note that also the average mixed strategy, or the best strategy D^t, constitutes an approximately optimal strategy as well, since its payoff is approximately the value of the game, against an optimally acting player.

Linear Programming and the Plotkin-Shmoys-Tardos framework

There are various ways in which MW theory can used to solve linear programs. Given what we developed in the previous section, one immediate way is to cast the LP as a zero-sum game and solve it via MW.

Note that there are some interesting trade offs between this idea and the traditional ways of solving linear programming problems. In particular, ellipsoid and interior point algorithms (IP) achieve an error of δ in $O(\text{poly}(n)\log(\frac{1}{\delta}))$ steps. Their dependence on the corresponding notion of the MW penalty width is logarithmic. On the other hand, the MW algorithm achieves an error after $O(\frac{\log(n)}{\delta^2})$ steps, in case the width is 1. Otherwise, the dependence on the width is quadratic, as we have shown. To summarize, IP algorithms are much better with respect to error and size of numbers (i.e., width), whereas MW are much better with respect to the dimension n.

We now switch focus to the Plotkin-Shmoys-Tardos framework, which is a more direct way of applying MW to linear programming. Our goal is to check to feasibility of a set of linear inequalities,

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

where $A = [a_1 \dots a_m]^T$ is an $m \times n$ matrix and x an n dimensional vector, or more precisely to find an approximately feasible solution $x^\star \geq 0$, such that for some $\delta > 0$,

$$a_i^T x^\star \geq b_i - \delta, \quad \forall i.$$

The analysis will be based on an oracle that answers the following question: Given a vector c and a scalar d, does there exist an $x \geq 0$, such that $c^T x \geq d$? With this oracle, we will be able to repeatedly check whether a convex combination of the initial linear inequalities, $a_i^T x \geq b_i$, is infeasible; a condition that is sufficient for the infeasibility of our original problem. Note that the oracle is straightforward to construct, as it involves a single inequality. In particular, it returns a negative answer if $d > 0$ and $c < 0$.

The algorithm is as follows. Experts correspond to each of the m constraints, and events correspond to points $x \geq 0$. The penalty for the ith expert for the event x will be $a_i^T x - b_i$, and is assumed to take values in $[-\rho, \rho]$. Although one might expect the penalty to be the violation of the constraint, it is exactly the opposite; the reason is that the algorithm is trying to actually prove infeasibility of the problem. In the tth round, we use our distribution over experts to generate an inequality that would be valid, if the problem were feasible: if our distribution is $p_1^t, \dots, p_m^t$, the inequality is $\sum_i p_i^t a_i^T x \geq \sum_i p_i^t b_i$. The oracle then either detects infeasibility of this constraint, in which case the original problem is infeasible, or returns a point $x^t \geq 0$ that satisfies the inequality. The penalty we pay then is equal to $\sum_i p_i^t (a_i^T x^t - b_i)$, and the weights are updated accordingly. Note that in case infeasibility is not detected, the penalty we pay is always nonnegative, since x^t satisfies the checked inequality.

If after $T = \frac{16\rho^2 \log(n)}{\delta^2}$ infeasibility is not detected, we have the following guarantee by the MW theory:

$$0 \leq \frac{\sum_{t=1}^T \sum_i p_i^t (a_i^T x^t - b_i)}{T} \leq \delta + \frac{\sum_{t=1}^T (a_i^T x^t - b_i)}{T},$$

for every i. The first inequality follows by the nonnegativity of all penalties. If we take $\bar{x}$ to be the average of all visited points x^t,

$$\bar{x} = \frac{\sum_t x^t}{T},$$

then this is our approximate solution, since from the above inequality we get for all i

$$0 \leq \delta + a_i^T \bar{x} - b_i \Rightarrow a_i^T \bar{x} \geq b_i - \delta.$$

Boosting

We now visit a problem from the area of Machine Learning. Suppose that we are given a sequence of training points, $x_1, \dots, x_N$, which are drawn from a fixed but unknown to us distribution $\mathcal{D}$. Alongide, we are given corresponding $0-1$ labels, $c(x_1), \dots, c(x_N)$, assigned to each point, where c is a function from some concept class $\mathcal{C}$ that maps points onto $0-1$ labels. Our goal is to generate a hypothesis function h that assigns labels to points, replicating the function c in the best way possible. This is captured by the average absolute error, $\mathbf{E}_{\mathcal{D}}[|h(x) - c(x)|]$. We call a learning algorithm to be *strong*, if for every distribution $\mathcal{D}$ and any

fixed $\epsilon, \delta > 0$, it outputs with probability at least $1 - \delta$ a hypothesis h that achieves error no more than ϵ. Similarly, it is called γ-*weak*, if the error is at most $0.5 - \gamma$.

Boosting is a very useful, both in theory and in practice, tool of combining weak rules of thumb into strong predictors. In particular, the theory of Boosting shows that if there exists a γ-weak learning algorithm for $\mathcal{C}$, then there also exists a strong one. We will show this in case we have a fixed training set with N points, and where the strong algorithm has a small error with respect to the uniform distribution on the training set.

We use the MW algorithm. In the tth round, we assign a different distribution $\mathcal{D}^t$ on the training set, and use the weak learning algorithm to retrieve a hypothesis h_t, which by assumption has error at most $0.5 - \gamma$, with respect to $\mathcal{D}^t$. Our final hypothesis after T rounds, h_{final}, is obtained by taking majority vote among $h_1, \ldots, h_T$. The experts in this case are the samples in the training set, and the events are the hypotheses produced by the weak learning algorithm. The associated penalty for expert x on hypothesis h_t is 1 if $h_t(x) = c(x)$, and 0 otherwise. As in the previous exemple, we penalize the experts that "are doing well", as we want to eventually increase the weight of a point (expert) if our hypothesis got it wrong. We can start with $\mathcal{D}^1$ being the uniform distribution, and we update according to the MW algorithm. Finally, after

$$T = \frac{2}{\gamma^2} \log \frac{1}{\epsilon}$$

rounds we get an error rate for h_{final} on the training set, under the uniform distribution, that is at most ϵ, as required.

Approximation Algorithms

We conclude with an application that demonstrates how to use the MW algorithm to get $O(\log n)$ approximation algorithms for many NP-hard problems. The problem that will focus on is the SET COVER problem: Given a universe of elements, $U = \{1, \ldots, n\}$, and a collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of subsets of U, whose union equals U, we want to pick a minimum number of sets from $\mathcal{C}$ to cover all of U. An immediate algorithm to tackle this problem is the greedy heuristic: at each step, choose the set from $\mathcal{C}$ that has not been chosen yet and that covers the most out of the yet uncovered elements of U. The MW algorithm will end up taking exaclty the form of that greedy algorithm, and will further prove the approximation bound.

We associate the elements of the universe with experts, and the sets of $\mathcal{C}$ with events. The penalty for expert i under event C_j, $M(i, C_j)$, will be equal to 1 if $i \in C_j$, and 0 otherwise. In this case, we use the following simplified rule for updating the weights,

$$w_i^{t+1} = w_i^t (1 - M(i, C_j)).$$

The update rule then gives elements that are covered by the newly chosen set a weight of 0, leaving the remaining unaltered. Consequently, the weight of element i in round t is either 0 or 1, depending on if it has being covered already, or not. The distribution we will be using then in round t,

$$p_i^t = \frac{w_i^t}{\sum_k w_k^t},$$

is just a uniform distribution over the uncovered elements by round t. We then choose the maximally adversarial event (that is, the one that maximizes the penalty), which coincides with the set C_j that covers a maximum number of uncovered elements, and update our weights. The described MW algorithm coincides with the greedy algorithm, in repeatedly picking the set that covers the most uncovered elements.

For any distribution $p_1^t, \ldots, p_n^t$ on the elements, we have that OPT sets cover everything. That means that the total weights of sets involved (accoring to the distribution p) is at least 1, and hence at least one of the remaining sets must cover at least 1/OPT fraction. Mathematically,

$$\max_j \sum_{i \in C_j} p_i^t \geq 1/\text{OPT}.$$

That shows that after every round, the total penalty drops significantly:

$$\Phi^{t+1} < \Phi^t e^{-1/\mathrm{OPT}}.$$

The inequality is strict, since the penalty is always positive. Using $\Phi^1 = n$, after OPT $\log n$ iterations we get $\Phi < 1 \Rightarrow \Phi = 0$, which shows that we can cover everything with OPT $\log n$ sets — an $\log n$ approximation.

18.409 Topics in Theoretical Computer Science: An Algorithmist's Toolkit
Fall 2009