Lecture 4

Applications of the experts algorithm

Let's go over some of the applications from the last lecture. You'll treat some of the others in your homework.

4.1 Finding linear classifiers

Let's get back to our first example, learning a linear classifier. For simplicity, we are going to assume that there exists a strict classifier, in the sense that there exists $x^* \in \mathbb{R}^N$ such that $\forall j = 1, ..., k,\ l_j(a_j \cdot x^*) \geq \varepsilon$ for some $\varepsilon > 0$ (assume x^* is normalized to have its elements sum to 1 and be a probability distribution). We also define $\rho = \max_j \|a_j\|_\infty$, where $\|a\|_\infty = \max_{i=1,...,N} |a_i|$.

The idea behind applying MWA here is to consider each of the N coordinates of vectors a_j as an expert, and choose the loss function $m^{(t)}$ as a function of one of the a_j that is not well classified by the current vector $p^{(t)}$. If $p^{(t)}$ is such that all of the a_j's satisfy $l_j(a_j \cdot p^{(t)}) \geq 0$, then all points are well classified by $x = p^{(t)}$ and we can stop the algorithm. Otherwise, we update $p^{(t)}$ through the MWA update rule and go to the next step. The question here is to determine why the algorithm terminates, and in how many steps.

Let us analyze the MW approach in a bit more details. We initialize $w_i^{(1)} = 1$ for all i, and $p^{(t)}$ accordingly. At each round t, we look for a value j such that $l_j(a_j \cdot p^{(t)}) < 0$, i.e. a_j is not classified correctly. If there is none, the algorithm stops. If there is one, pick this j and define $m^{(t)} = -\frac{l_j}{\rho} a_j$; note that by definition of ρ, $m_i^{(t)} \in [-1, 1]$ for all i.

By assumption, there exists x^* such that

$$m^{(t)} \cdot x^* = -l_j(a_j \cdot x^*)/\rho \leq -\varepsilon/\rho.$$

Using that $\eta \geq 0$, $|m_i^{(t)}| \leq 1\ \forall i$, and $\sum_{i=1}^{N} x_i^* = 1$, we also have

$$\eta |m^{(t)}|^2 \cdot p \leq \eta,$$

for any distribution vector p. By the main MW theorem, since x^* is a distribution, we have

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq \sum_{t=1}^{T} m^{(t)} \cdot x^* + \frac{\ln N}{\eta} + \eta T.$$

Using $m^{(t)} \cdot x^* \leq -\varepsilon/\rho$, we get

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq -\frac{\varepsilon}{\rho} T + \frac{\ln N}{\eta} + \eta T.$$

Now, one should remark that until the last time step T, i.e. until the algorithm stops, we chose a_j such that $l_j(a_j \cdot p^{(t)}) < 0$, which implies that $\forall t = 1, ..., T$, $m^{(t)} \cdot p^{(t)} = -l_j(a_j \cdot p^{(t)})/\rho > 0$. It immediately follows that

$$0 < -\frac{\varepsilon}{\rho} T + \frac{\ln N}{\eta} + \eta T.$$

Finally, choosing $\eta = \frac{\varepsilon}{2\rho}$, we obtain $T < \frac{4\rho^2 \ln N}{\varepsilon^2}$. This proves that the algorithm terminates, and that it finds a classifier in less than $\frac{4\rho^2 \ln N}{\varepsilon^2}$ timesteps.

4.2 Zero-sum games

Next we consider games in the game theoretic sense: informally, a set of k players each have a set of a actions they can choose from; every player chooses an action, after which he gets a payoff or utility that is a function of his own action and the actions of other players. Here will limit ourselves to 2-player zero-sum games, in which each of the two players has a finite number N of available actions. The payoffs of players 1 and 2 (let us call them the row and column players from now on) can be represented by a single payoff matrix $M \in \mathbb{R}^{N \times N}$ such if the row player plays action i, and the column player plays j, then the column player receives payoff $M(i, j)$ and the row player receives $-M(i, j)$.

A player can decide to randomize and choose a distribution over actions which she will sample from. In particular, if the row player plays according to a distribution p over row actions (instead of playing one specific action) and the column player plays according to a distribution q over column actions, then the expected payoff of the column player is $p^T M q$ and the expected payoff of the row player is $-p^T M q$.

Given that each player is trying to maximize her utility, we can see that the row and column players in a zero-sum games have conflicting objectives: the column player wants to maximize $p^T M q$, while the row player want to minimize the same quantity. One nice property of zero-sum games is that it does not matter for any of the players whether they act first: Von Neumann's minimax theorem states that

$$\max_q \min_p p^T M q = \min_p \max_q p^T M q = \lambda^*.$$

This means in particular that it does not matter whether the row player tries to minimize $p^T Mq$ first and lets the column player act second and maximize $\min_p p^T Mq$ or the column player tries to maximize $p^T Mq$ first and the row player then minimizes $\max_q p^T Mq$: the utility both players end up getting does not change. This "optimal" utility λ^* is called the value of the game.

We want to show that the MWA allows us to find strategies that are near-optimal for a zero-sum game, in the sense that for any $\varepsilon > 0$ and for a sufficient number of time steps T, the MWA finds a set of strategies $(\frac{1}{T}\sum_{t=1}^T p^{(t)}, \frac{1}{T}\sum_{t=1}^T q^{(t)})$ for the row and column players that satisfy

$$\lambda^* \leq \left(\frac{1}{T}\sum_{t=1}^T p^{(t)}\right)^T M \left(\frac{1}{T}\sum_{t=1}^T q^{(t)}\right) \leq \lambda^* + \varepsilon.$$

The idea behind the MW here is to take $p^{(t)}$ to be the strategy of the row player. At each time step, $q^{(t)}$ is chosen as the best response to strategy $p^{(t)}$ (i.e. the strategy which maximize the utility of the column player given that the row player plays $p^{(t)}$); then, the row player uses the multiplicative weight update based on a loss vector $m^{(t)}$ chosen as the vector of expected payoffs for the row player when the column player plays $q^{(t)}$ to decide what $p^{(t+1)}$ will be in the next time step. Intuitively, the row player tries to penalize the strategies that lead to a low payoff and to put more weight on the strategy that leads to a higher payoff. Formally, the algorithm proceeds as follows:

Algorithm 1: MWA for zero-sum games

Choose $\eta \in (0, 1/2]$;
Initialize weights $w_i^{(1)} := 1$ for each $i \in \{1, \ldots, N\}$;
for $t = 1, 2, \ldots, T$ **do**
  Define the distribution $p^{(t)} = \{\frac{w_1^{(t)}}{\Phi^{(t)}}, \ldots, \frac{w_N^{(t)}}{\Phi^{(t)}}\}$, where $\Phi^{(t)} = \sum_i w_i^{(t)}$;
  Sample an action i from distribution $p^{(t)}$;
  Define $q^{(t)} = \text{argmax}_{q^{(t)}} (p^{(T)})^T Mq^{(T)}$ and $m^{(t)} = Mq^{(t)}$;
  Penalize costly decisions by updating their weights: $w_i^{(t+1)} \leftarrow e^{-\eta m_i^{(t)}} w_i^{(t)}$;
end

Let's analyze this algorithm. Assume that the payoffs $M(i,j) \in [-1, 1]$. Choosing $\eta = \sqrt{\ln N/T}$ and $T = 4 \ln N/\varepsilon^2$, according to the MW theorem, for any distribution p^*:

$$\frac{1}{T}\sum_{t=1}^T m^{(t)} \cdot p^{(t)} = \frac{1}{T}\sum_{t=1}^T (p^{(t)})^T Mq^{(t)} \leq \frac{1}{T}\sum_{t=1}^T (p^*)^T Mq^{(t)} + \varepsilon.$$

Take $p^* = \text{argmin}_p p^T Mq^{(t)}$. Then for any t

$$(p^*)^T Mq^{(t)} \leq \min_p p^T Mq^{(t)} \leq \max_q \min_p p' Bq$$

and therefore

$$(p^*)^T M q^{(t)} \leq \max_q \min_p p^T M q = \lambda^*,$$

so that

$$\frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T M q^{(t)} \leq \lambda^* + \varepsilon.$$

To lower bound $\frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T M q^{(t)}$, note that for every distribution q,

$$(p^{(t)})^T M q^{(t)} \geq (p^{(t)})^T M q$$

given the choice of $q^{(t)}$ made in the algorithm; therefore,

$$(p^{(t)})^T M q^{(t)} \geq \max_q (p^{(t)})^T M q$$

and

$$(p^{(t)})^T M q^{(t)} \geq \min_p \max_q p^T M q = \lambda^*.$$

Using the fact that the above bounds are valid for any t, by linearity we get that the distributions produced by the algorithm satisfy

$$\lambda^* \leq \left(\frac{1}{T}\sum_{t=1}^{T} p^{(t)}\right)^T M \left(\frac{1}{T}\sum_{t=1}^{T} q^{(t)}\right) \leq \lambda^* + \varepsilon.$$

This equation shows that p^* and q^* are approximate equilibria of the game, in the sense that neither the row or the column player can improve her utility by more than ε by changing her strategy, when the strategy of the other player remains fixed. Indeed, imagine that the column player decides to change her strategy to another q. Then, she is going to get payoff

$$\left(\frac{1}{T}\sum_{t=1}^{T} p^{(t)}\right)^T Mq \leq \frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T M q^{(t)} \leq \lambda^* + \varepsilon.$$

However, she is guaranteed to earn payoff at least λ^* when playing q^*, and can therefore not improve her payoff by more than ε by changing her strategy. A similar reasoning applies to the row player.

4.3 Solving linear programs

Our goal is to check to feasibility of a set of linear inequalities,

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

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

$$a_i^T x^* \geq b_i - \varepsilon, \qquad \forall i.$$

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 loss functions are associated with points $x \geq 0$. The loss suffered by the i-th expert will be $m_i = \frac{1}{\rho}(a_i^T x - b_i)$, where $\rho > 0$ is a parameter used to ensure that the losses $m_i \in [-1, 1]$. (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 t-th round, we use our distribution $p^{(t)}$ over experts to generate an inequality that would be valid, if the problem were feasible: the inequality is

$$\sum_i p_i^{(t)} a_i^T x \geq \sum_i p_i^{(t)} b_i. \tag{4.1}$$

We then use the oracle to detect whether this constraint is infeasible, in which case the original problem is infeasible, or return a point $x^{(t)} \geq 0$ that satisfies the inequality. The loss we suffer is equal to $\frac{1}{\rho} \sum_i p_i^{(t)} (a_i^T x^{(t)} - b_i)$, and we use this loss to update our weights. Note that in case infeasibility is not detected, the penalty we pay is always nonnegative, since $x^{(t)}$ satisfies the inequality (4.1).

The simplified form of the multiplicative weights theorem gives us the following guarantee:

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} \cdot p_i^{(t)} + 2\rho \sqrt{\frac{\ln m}{T}},$$

for any i. Choose $T = 4\rho^2 \ln(m)/\varepsilon^2$. Using that the left-hand side is non-negative, if we set

$$x^* = \frac{1}{T} \sum_{t=1}^{T} x^{(t)}$$

we see that for every i, $a_i^T x^* \geq b_i - \varepsilon$, as desired.

The number of iterations required to decide approximate feasibility, T, scales logarithmically with the number of constraints m. This is much better than "standard" algorithms such as the ellipsoid algorithm, which are polynomial in n. However, the dependence on the accuracy ε is quadratic, which is much worse than these algorithms, for which the dependence is in $\log(1/\varepsilon)$. Finally the dependence of T on ρ^{-1} is also quadratic. The value of ρ depends on the oracle (it is called the "width" of the oracle), and depending on the specific LP we are trying to solve it may or may not be possible to design an oracle that guarantees a small value of ρ.