CS 170 HW 12

Due on 2018-04-23, at 11:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (—) Nostalgia

What's been your favorite homework problem this semester? Tell us the HW number and problem name/number, and briefly explain (a sentence or two) why you liked it.

3 (★★) Multiplicative Weights

(a) We are running Multiplicative Weights with $n = 2$ experts for $T = 10,000$ days, and we choose $\epsilon = 0.01$ Is this close to the optimum choice? (Hint: $\sqrt{\ln 2} \approx .8325$)

(b) In all of the first 140 days, Expert 1 has cost 0 and Expert 2 has cost 1. In the next day, with what probability will you play Expert 1? (Hint: You can assume that $0.99^{70} = \frac{1}{2}$)

Solution:

(a) We need $\sqrt{\frac{\ln n}{T}} \leq \epsilon$. Plugging in what we know, we have $(\sqrt{\ln 2}/100) \approx .008325$. This is close enough.

(b) The weight assigned to expert 1 is $.99^{0 \cdot 140} = 1$, while the weight assigned to expert 2 is $.99^{1 \cdot 140} \approx 1/4$. So, the probability we play expert 1 is $\frac{1}{1+1/4} = 4/5$.

4 (★★★★) Experts Alternatives

Recall the experts problem. Every day you must take the advice of one of n experts. At the end of each day t, if you take advice from expert i, the advice costs you some c_i^t in $[0, 1]$. You want to minimize regret R,

$$R = \frac{1}{T}(\sum_{t=1}^{T} c_{i(t)}^t - \min_{i \leq i \leq n} \sum_{t=1}^{T} c_i^t)$$

where $i(t)$ is the expert you choose on day t. Your strategy will be probabilities where p_i^t denotes the probability with which you choose expert i on day t. Assume an all powerful adversary can look at your strategy ahead of time and decide the costs associated with each expert on each day. Give the maximum possible expected regret that the adversary can guarantee in each of the following cases:

(a) Choose expert 1 at every step. That is, if $\forall t\ p_1^t = 1$, what is $\max_{C_i^t} R$? C_i^t is the set of costs for all experts and all days.

(b) Any deterministic strategy. Note that a "determinstic strategy" can be thought of as a probability distribution that satisfies the following: $\forall t\ \exists i\ p_i^t = 1$.

(c) Always choose an expert according to some fixed probability distribution at every time step. That is, if for some $p_1 ... p_n$, $\forall t, p_i^t = p_i$, what is $\max_{C_i^t}(\mathbb{E}[R])$?)

(d) In the last case, express your worst-case regret bound in terms of the $p_i's$, where $p_i \geq 0$ is the probability of choosing expert i and $\sum_i p_i = 1$. What is the best distribution? In other words, what is $\text{argmin}_{p_1 ... p_n} \max_{C_i^t}(\mathbb{E}[R])$?)

This analysis should conclude that a good strategy for the problem must necessarily be randomized and adaptive.

Solution:

a Answer: 1. Consider the case where the cost of expert 1 is always 1 and the cost of expert 2 is always 0. Thus $C = \sum_{t=1}^{T} \sum_{i=1}^{n} p_i^t c_i^t = \sum_{t=1}^{T} c_1^t = T$, we have $C^* = \sum_{i=1}^{T} c_2^t = 0$, so regret is $\frac{1}{T}(T - 0) = 1$.

b Answer: $\frac{n-1}{n}$. Consider the case where the cost of the chosen expert is always 1, and the cost of each other expert is 0. Let k be the least-frequently chosen expert, and let m_k be the number of times that expert is chosen. This will result in a regret of $\frac{1}{T}(T - m_k)$

Since the best expert is the one that is chosen least often, the best strategy will try to maximize the number of times we choose the expert that is chosen least often. This means we want to choose all the experts equally many times, so expert k is chosen in at most T/n of the rounds. Therefore, $m_k \leq \frac{T}{n}$, thus the regret is at least $\frac{1}{T}(T - \frac{T}{n}) = \frac{n-1}{n}$

c Answer: $(1 - \min_i p_i)T$. Like in part 1, the distribution is fixed across all days, so we know ahead of time which expert will be chosen least often in expectation. Let $k = \arg\min_i p_i$ be the expert with least cost. Let $c_k^t = 0$ for all t, and let $c_i^t = 1$ for all $i \neq k$ and for all t. This way, $C = T(1-p_k)$, as this is a binomial random variable distributed as $C \sim Bin(T, 1-p_k)$. $C^* = 0$, so we end up with a regret of $\frac{1}{T}(C - C^*) = \frac{1}{T}(T(1 - p_k) - 0) = 1 - p_k$.

d Answer: From part c), we know that regret $= (1 - \min_i p_i)$. Thus, to minimize this expression is the same as maximizing $\min_i p_i$. This is maximized by the uniform distribution, obtaining regret $\frac{n-1}{n}$ (this is the same worst case regret as in part 2).

5 (★★★★) [OPTIONAL] Knightmare

Give an algorithm to find the number of ways you can place knights on an N by M chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights). The runtime should be $O(2^{3M} \cdot N)$ (or symmetrically, switch the variables).

Solution:

Pseudocode:

Initialize $K(\cdot, \cdot, \cdot) := 0$;
for *all size* M *bitstrings* v, w **do**
  Initialize $K(2, v, w) := 1$ if v, w is valid else 0;
end
for $n = 3$ *to* N **do**
  for *all size* M *bitstrings* u, v, w *if* u, v, w *is valid* **do**
    $K(n, v, w)$ += $K(n-1, w, u)$;
  end
end
return $\sum_{v,w} K(N, v, w)$;

Proof of Correctness: By definition, $K(2, v, w) = 1$ if the M by 2 chessboard configuration defined by v and w is legitimate. Otherwise, $K(2, v, w) = 0$.

For $n > 2$, we have

$$K(n, v, w) = \sum_u K(n-1, w, u),$$

where we are summing over all possible configurations u for the third-last column of a chessboard whose last columns are specified by w and v.

We can precompute all valid M by 2 configurations and M by 3 configurations in $O(2^{3M}M)$ time because we can just check that each square with a knight is not being attacked by any other square with a knight. A square can only be under attack by at most 8 other squares so this check takes constant time.

6 (★★★★) [OPTIONAL] Min Cost Flow

In the max flow problem, we just wanted to see how much flow we could send between a source and a sink. But in general, we would like to model the fact that shipping flow takes money. More precisely, we are given a directed graph G with source s, sink t, costs l_e, capacities c_e, and a flow value F. We want to find a nonnegative flow f with minimum cost, that is $\sum_e l_e f_e$, that respects the capacities and ships F units of flow from s to t.

(a) Show that the minimum cost flow problem can be solved in polynomial time.

(b) Show that the shortest path problem can be solved using the minimum cost flow problem

(c) Show that the maximum flow problem can be solved using the minimum cost flow problem.

Solution:

(a) Write min-cost flow as a linear program. One formulation is as follows:

$$\begin{aligned} \min & \sum_e l_e f_e \\ \text{subject to } & 0 \le f_e \le c_e && \forall e \in E \\ & \sum_{e \text{ incoming to } v} f_e = \sum_{e \text{ outgoing from } v} f_e && \forall v \in V, v \neq s, t \\ F + & \sum_{e \text{ incoming to } s} f_e = \sum_{e \text{ outgoing from } s} f_e \end{aligned}$$

(b) Consider a shortest path instance on a graph G with start s, end t, and edge weights l_e. Let $F = 1$ and let $c_e = 1$ for all edges e. We will now show that in the min cost flow f, the subgraph H of G consisting of edges with nonzero flow, all directed paths from s to t have length equal to the shortest path from s to t. Therefore, to find a shortest path from s to t, it is enough to use a DFS or BFS to find any path between s to t in H.

Suppose that each $l_e > 0$ (we can contract any edge with $l_e = 0$). First, we show that H is a DAG. If H is not a DAG, it must contain a cycle. Let $\delta > 0$ be the minimum flow along any edge of this cycle. Subtracting δ from the flow of all edges in this cycle results in a flow that still ships 1 unit of flow from s to t. This flow, though, has smaller cost than before, a contradiction to the fact that f is a min-cost flow. Therefore, H is a DAG.

Consider any path P between s and t in H. Let δ be the minimum flow on any edge of this path. One can write the flow f as a linear combination of paths including P by writing

$$f = \delta p + \delta_1 q_1 + \ldots + \delta_k q_k$$

where p and q_i are the unit flows along the paths P and Q_i respectively, δ_i is the minimum amount of flow on some arbitrary path Q_i after subtracting flow from $P, Q_1, Q_2, \ldots, Q_{i-1}$. Since subtracting δ_i units of flow decreased the flow from s to t by δ_i units, $\delta + \sum_i \delta_i = 1$. Since the objective function for min-cost flow is linear, we can write

$$\sum_e l_e f_e = \delta \sum_e l_e p_e + \sum_i \delta_i \sum_e l_e q_{ie} = \delta \sum_{e \in P} l_e + \sum_i \delta_i \sum_{e \in P_i} l_e$$

In particular, the cost of the flow f is the average length of the paths $P, Q_1, \ldots, Q_k$ according to the weights $\delta, \delta_1, \ldots, \delta_k$. Therefore, if P or some Q_i is not a shortest path from s to t, one can decrease the cost of f by reassigning the flow along that path to a shortest path between s and t. This means that if f has minimum cost, P and all Q_is must be shortest paths between s and t. Since P was chosen to be an arbitrary path from s to t in H, all paths in H must be shortest paths between s and t.

(c) We can use binary search to find the true max flow value. Consider a max flow instance on a graph G with capacities c_e where we wish to ship flow from s to t. Create a min-cost flow instance as follows. Set the capacities to be equal to c_e and let the lengths be arbitrary (say $l_e = 1$ for all edges). We know that the max flow value $F_{max} \leq \sum_e c_e$. If the capacities are integers, F_{max} is also an integer. Therefore, we can binary search to find its true value. For an arbitrary F, we can find out if there is a flow that ships more than F units of flow by querying our min-cost flow instance with value at least F. If there is a flow with value at least F, it will return a flow with finite cost. Otherwise, the program is infeasible.

7 (★★★★) [OPTIONAL] Graph Game

Given an undirected, unweighted graph, with each node having a certain value, consider the following game.

1. All nodes are initially *unmarked* and your score is 0.
2. First, you choose an unmarked node u. You look at the neighbors of u and add to your score the sum of the values of the *marked* neighbors v of u.
3. Then, mark u.
4. You repeat the last two steps for as many turns as you like (you *do not* have to mark all the nodes. Each node can be marked at most once).

For instance, suppose we had the graph $A - B - C$ with A, B, C having values $3, 2, 3$ respectively. Then, the optimal strategy is to mark A then C then B giving you a score of $0 + 0 + 6$. We can check that no other order will give us a better score.

a Suppose all the node values are nonnegative. Give an efficient greedy algorithm to determine the order to mark nodes to maximize your score. Justify your answer with an exchange argument.

b Now, node values can be negative. Show this problem is NP-hard by giving a reduction from INDEPENDENT SET

Solution:

a We claim that sorting the nodes by decreasing value (breaking ties arbitrarily), and taking them in that order will lead to an optimal solution. We show this by an exchange argument. For any other ordering π that isn't sorted, we can find an i such that $l(\pi_i) < l(\pi_{i+1})$. Now, suppose we swapped them. We have two cases:

(a) Case 1: π_i and π_{i+1} do not have an edge between them: In this case, swapping these two nodes will not decrease our score, since these are unrelated nodes.

(b) Case 2: π_i and π_{i+1} do have an edge between them: We can ignore all neighboring nodes, since we will mark these two nodes eventually. However, notice that it is optimal to mark the higher valued node first, then the lower valued one, thus, swapping these will give us a strictly better score.

Thus, by the exchange argument, sorting the nodes will give us an optimal solution.

b An instance of independent set consists of a graph $G = (V, E)$ and a number b, which means we wish to find an independent set of size at least b. We transform it to this problem as follows: for each of the original nodes in the graph, assign it a value of -2. Add a "super-node" s and connect it to everything else, and assign it a value of 1. We also set our target value to be b. We now claim that our score directly corresponds to maximizing the size of an independent set

(a) Solution to independent set to solution to graph marking: If there exists an independent set of size b, then our strategy for graph marking would be to mark s, then the nodes in the independent set, giving us a score of b.

(b) Solution to graph marking to solution to independent set: First, notice that it is always optimal to mark s first regardless. Now, suppose we've gotten a score of b. Now, if we have chosen any node that neighbors a marked node that is not s, we will receive a net score of -1, which is clearly not optimal. Thus, we can see that graph marking will never choose any two nodes that are adjacent. Thus, this directly corresponds to an independent set of size b.

Thus, we've completed both directions, so we've shown that graph marking is NP-hard.