15-750: Algorithms in the Real World **CMU, Spring 2023**
HW #5 (out Tuesday March 28, 2023) **Due:** Tuesday April 11, 2023

Please limit yourself to groups of two (or three) people. We adhere to a "whiteboard" collaboration policy: you should discuss in your group on whiteboards (or equivalents) but then should go away and write down your solutions yourself. You must not share written work. **You must write down the names of the person(s) with whom you collaborate.** In general, you should solve the homework problems using only to material we refer you to (or put on the course page), and not books or other online resources. If you have reason to use any resources we point you to (except the course notes), please cite them.

A word of advice: please try to solve the problem by yourself before working with your group. That is the way to improve your problem-solving skills.

Homeworks will be due at 11:59pm on the due date on `gradescope`. Corrections and changes will appear on the course webpage and on Piazza; please check them regularly.

Problems

1. Given a set of points in the 2D plane, there are several ways to perform "regression", meaning trying to fit a line $y = mx + b$ to the points. Generally you want to minimize the "error" (badness of fit), but this depends on how you define "error".

   Suppose $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ and $\vec{y} = \{y_1, y_2, \ldots, y_n\}$ are the x- and y-coordinates of the set of points (so (x_i, y_i) is the coordinate of the ith point). One possible model is "L_2 regression", in which the goal is to find m, b minimizing

   $$\text{err}_2(m, b) = \sum_{i=1}^{n} (y_i - mx_i - b)^2.$$

   In this case, there is simply a closed-form solution for the best m, b; it's literally

   $$m = \frac{(\sum_{i=1}^{n} x_i y_i) - n\bar{x}\bar{y}}{(\sum_{i=1}^{n} x_i^2) - n\bar{x}^2},$$
   $$b = \bar{y} - m\bar{x},$$

   where $\bar{x}$ denotes the average of the x_i's (similarly $\bar{y}$). You might enjoy proving that, but you don't have to. The fact that this closed form exists is the main reason L_2 regression is popular.

   Two more possibilities are "L_1 regression" and "L_∞ regression". In these, the goal is to find m, b minimizing

   $$\text{err}_1(m, b) = \sum_{i=1}^{n} |y_i - mx_i - b|$$

   and

   $$\text{err}_\infty(m, b) = \max_{i=1}^{n} |y_i - mx_i - b|,$$

   respectively. For each of these two problems, L_1 and L_∞ regression, show how to find the optimum m, b using a linear program with at most $O(n)$ variables and constraints. In both cases, it is sufficient to write a correct LP and give a short (two or three sentence) explanation of why it gives the optimal result.

2. Consider the following task: The input is a list of subsets $S_1, S_2, \ldots, S_m \subseteq \{1, 2, 3, \ldots, n\}$, and also a positive integer $k \leq m$. The challenge is to pick k of the subsets so that their union is as large as possible; that is, we want to find a list $L = (i_1, i_2, \ldots, i_k)$ of numbers (between 1 and m) so that $|\mathcal{U}_L|$ is as large as possible, where

$$\mathcal{U}_L = S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_k}.$$

(Technically, the indices $i_1, i_2, \ldots, i_k$ in L do not have to be distinct, although it can never hurt to make them distinct.)

This task is NP-hard! So we don't expect any polynomial-time algorithm can always find the optimal solution. In this problem we'll develop an efficient randomized algorithm that gives a good "approximation ratio".

(a) Given an input instance, consider the following linear program (LP) with variables $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} y_j \\
\text{subject to} \quad & \sum_{i=1}^{m} x_i = k \\
& \sum_{i \text{ such that } j \in S_i} x_i \geq y_j \quad \text{for all } 1 \leq j \leq n \\
& 0 \leq x_i \leq 1 \quad \text{for all } 1 \leq i \leq m \\
& 0 \leq y_j \leq 1 \quad \text{for all } 1 \leq j \leq n
\end{aligned}$$

Suppose we turn the above into an integer linear program (ILP) by adding the constraint that the x_i's and y_j's all have to be integers. Explain why the resulting ILP *exactly* corresponds to the optimal solution. (Of course, we aren't able to solve ILPs in polynomial time!) Add a sentence reminding the reader why this means that if the LP's optimum value is α, and the true optimum value is β, then $\alpha \geq \beta$.

(b) Suppose we solve the above LP (in polynomial time). Of course, the resulting x_i and y_j values need not be integers. Let $p_i = x_i/k$ for each $1 \leq i \leq m$; note that these values form a probability distribution, because they are nonnegative and they must add up to 1 (by the first LP constraint). Suppose we pick a random index i_1 according to this distribution (meaning the probability of i_1 being i is p_i). Show that, for any j between 1 and n, the probability that S_{i_1} contains j is at least y_j/k.

(c) Consider the algorithm that — after solving the LP — produces a candidate list L by picking k indices $i_1, \ldots, i_k$ independently at random from the probability distribution defined by the p_i's. Explain why for each j, we have $\mathbf{Pr}[j \in \mathcal{U}_L] \geq 1 - (1 - y_j/k)^k$.

(d) Put yourself in the mindset of calculus. Defining $f(y) = 1 - (1 - y/k)^k$ and $g(y) = (1 - 1/e)y$, prove that $f(0) \geq g(0)$ and $f(1) \geq g(1)$ (you will need that $e^{-t} \geq 1 - t$ for all t). In fact, $f(y) \geq g(y)$ for all $0 \leq y \leq 1$. (If you really enjoy calculus, you can finish proving that — hint, show that f is concave. You might also enjoy "proving" it to yourself by plotting the two functions. But I don't insist on this.)

(e) Deduce that the expected value of $|\mathcal{U}_L|$ produced by this randomized rounding algorithm is at least $(1 - 1/e)\alpha \geq (1 - 1/e)\beta$. (This means this algorithm gets $1 - 1/e \approx 63\%$ of the optimal solution, in expectation.)

3. The eternal dilemma is: buy now, or rent for a little longer? Assume it costs $\$B$ to buy skis, and \$1 per day to rent. In class we saw a deterministic algorithm with competitive ratio $2 - 1/B$. Let's investigate the optimal randomized algorithm.

   Suppose you develop any kind of randomized algorithm for deciding each day whether to rent or buy. You might imagine simulating this algorithm a bunch of times; every time you simulate it, you end up buying on some certain day J (or possibly never). There is some probability J will be 1, some probability J will be 2, etc. So in a sense, your whole algorithm is defined by a list of probabilities $p_1, p_2, p_3, \ldots$ that add up to 1, and the meaning is that with probability p_J you do

   "the day-J strategy": rent for the first $J-1$ days, and buy on the Jth day.

   On the other hand, the possible number of days you end up skiing is any integer $I \geq 1$. You might think of the Devil as selecting this number I.

   Recall that once you fix an algorithm — meaning probabilities $p_1, p_2, p_3, \ldots$ — we are interested in its competitive ratio, which is

$$\max_I \left\{ \frac{\mathbf{E}[\text{cost of algorithm for } I \text{ days of skiing}]}{\text{optimal cost for } I \text{ days of skiing}} \right\}.$$

   You can think of there being a 2-player game between you and the Devil. First, you commit to using the randomized algorithm defined by $p_1, p_2, p_3, \ldots$. Then the Devil selects the worst I. Then we study your expected cost for I days, divided by the optimal cost for I days. Here your expected cost is p_1 times the cost of using the "day-1 strategy", plus p_2 times the cost of using the "day-2 strategy", plus p_3 times the cost of using the "day-3 strategy", etc.

   (a) Given positive integers I, J, define the "pain" R_{IJ} to be this ratio: the cost of doing the day-J strategy for I days of skiing, divided by the optimal cost for I days of skiing. Let R be the "pain" matrix, whose rows I correspond to the Devil's possible choices, and whose columns J correspond to the different day-J strategies. Write down the expression for R_{IJ}.

   (b) This matrix has an infinite number of row and columns, which is annoying. Let's add one more row, called "row ∞" for the scenario that we ski for all eternity. Argue that without loss of generality, we can assume the Devil chooses only values $I \in \{1, 2, \ldots, B-1\}$ or $I = \infty$. Formally, argue that for any finite $I \geq B$ we have $R_{IJ} \leq R_{\infty J}$ for all J.

   (c) Now that the matrix has just B rows, argue that it's never in your best interest to use the day-J strategy for any $J > B$. Formally, argue that for any $J > B$ we have $R_{IJ} \geq R_{IB}$ for $I \in \{1, 2, \ldots, B-1, \infty\}$.

   (d) Thus we may as well only consider a $B \times B$ pain matrix. Write down this matrix for $B = 2, 3, 4$, and then give a formula for R_{IJ} in the general case.

4. We now continue on to investigate the optimal randomized algorithm!

   (a) Suppose we fix a particular B, which in turn fixes the $B \times B$ pain matrix R. Recall that the randomized algorithm is defined by the probabilities $p_1, p_2, \ldots, p_B$. Show that there is a linear program with either B or $B+1$ variables (depends a little how you choose to do it) whose solution yields the randomized algorithm with smallest competitive ration for this particular B.

(b) Determine the optimal competitive ratio for a randomized algorithm in the case $B = 2$. (The answer should be a nice number smaller than $2 - 1/B = 3/2$.)

(c) (BONUS, up to 3 points.) Get a computer program to set up and solve the LP for various values of B. Explain how you do this, and show some results. Maybe plot the optimal competitive ratio for increasing values of B and compare against $\frac{e}{e-1}$. Maybe plot the optimal p_J values for various values of B. Maybe try guessing — for each B — a set of p_J values that works "pretty well" (though guessing an exact formula for the *optimal* p_J's is pretty hard!). I was able to guess a formula for some p_J values (for each B) that let me show the optimal competitive ratio is no worse than $5/3$. Can you do the same? See what you can work out!