CS218 Design and analysis of algorithms **Jan-Apr 2025**

Exercises: Linear Programming, Approximation, Randomized, Error correction

1. (Line fitting.) Given n points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, with labels $\ell_1, \ell_2, \ldots, \ell_n \in \mathbb{R}$, we want to compute a linear function that best fits with the points and labels. More precisely, find a function $h(x) = a_1x_1 + a_2x_2 + \cdots + a_dx_d + b$ so that we minimize the error function $E(h)$ defined as

$$E(h) = \max_{1 \le j \le n} \{|h(p_j) - \ell_j|\}.$$

   Write a linear program for this.

2. (Curve fitting.) Given n points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, with labels $\ell_1, \ell_2, \ldots, \ell_n \in \mathbb{R}$, we want to compute a quadratic function that best fits with the points and labels. More precisely, find a function $h(x) = \sum_{1 \le i \le j \le d} a_{i,j}x_ix_j$ so that we minimize the error function $E(h)$ defined as

$$E(h) = \max_{1 \le j \le n} \{|h(p_j) - \ell_j|\}.$$

   Write a linear program for finding h.

3. (Classification.) Given n points $p_1, p_2, \ldots, p_n \mathbb{R}^d$, which are labeled either positive or negative, we want to compute a linear function that best fits with the points and labels. More precisely, find a function $h(x) = a_1x_1 + a_2x_2 + \cdots + a_dx_d + b$ so that we minimize the hing loss $L(h)$ defined as follows.

   For a point p_j, if it's label is positive then we expect $h(p_j)$ to be (significantly) more than zero. Let's say we expect $h(p_j)$ to be at least 1. If that is not true then we consider the difference from 1 as the loss. Define loss with respect to a positively labeled point p_j as

$$L(h, p_j) \begin{cases} = 1 - h(p_j) & \text{if } h(p_j) < 1 \\ = 0 & \text{otherwise.} \end{cases}$$

   Similarly, define loss with respect to a negatively labeled point p_k as

$$L(h, p_k) \begin{cases} = h(p_k) + 1 & \text{if } h(p_j) > -1 \\ = 0 & \text{otherwise.} \end{cases}$$

   Finally we define the hinge loss $L(h)$ over all points as $\sum_j L(h, p_j)$. Write a linear program to find h that minimizes $L(h)$.

4. (Simplex algorithm simulation) Consider the following linear program.

$$\begin{aligned} \max 2x_1 + x_2 \text{ subject to} \\ x_1 &\ge 0 \\ x_2 &\ge 0 \\ x_1 &\le 1 \\ x_1 + x_2 &\le 2 \end{aligned}$$

   - First write the LP in the standard form where we have (two) equations and (four) non-negativity constraints $x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0$.

- Start with the basic feasible solution that has $x_1 = 0, x_2 = 0$. What will be the values of x_3, x_4? At any point in the algorithm, the two variables which are zero are called the *non-basic variables* and the remaining are called *basic variables*. For example, in the beginning, x_3, x_4 are basic variables and x_1, x_2 are non-basic.
- Run the iterations of the simplex algorithm till you get an optimal solution. After each iteration write down (i) the basic feasible solution and (ii) express the objective function in terms of the current non-basic variables (i.e., which are zero). In an iteration, there may be multiple possible choices for which variable to increase. You can just make an arbitrary choice.

Below is how an iteration of simplex algorithm runs.

- Choose one of the non-basic variables to increase. It should be among those whose coefficient in the objective function is positive. If there is no such variable then output the current solution. If there is such a variable then increase it till the maximum possible value while maintaining feasibility. The other non-basic variables (except the chosen one) should remain zero in this iteration.
- This gives you a new basic feasible solution and a new set of basic and non-basic variables
- Express the objective function in terms of the non-basic variables.
- Express the basic variables in terms on non-basic variables.

5. Prove that when the simplex algorithm stops, that is, when we express the objective function in terms of non-basic variables and all coefficients turn out to be negative, then we are at an optimal solution. What is the optimal value at this point?

6. (Initial basic feasible solution.) We said that finding initial basic feasible solution itself is a challenging problem. We will solve it by framing it as another linear program. Suppose the given linear program (LP1) is

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ &\vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \cdots + a_{k,n}x_n &= b_k \\ x_1, x_2, \ldots, x_n &\geq 0. \end{aligned}$$

Here $a_{i,j}$s and b_is are given as input and x_js are unknowns. Without loss of generality, we can assume that $b_1, b_2, \ldots, b_k \geq 0$. Because otherwise we can simply multiply -1 on both the sides of the equation.

We write the following new linear program (LP2).

$$\begin{aligned} \min z_1 + z_2 + \cdots + z_k \text{ subject to} \\ a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n - b_1 &= z_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n - b_2 &= z_2 \\ &\vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \cdots + a_{k,n}x_n - b_k &= z_k \\ x_1, x_2, \ldots, x_n &\geq 0. \\ z_1, z_2, \ldots, z_k &\geq 0. \end{aligned}$$

Here z_is and x_js are unknowns.

Prove that LP1 has a feasible solution if and only if LP2 has an optimal value =0.

There is a trivial initial basic feasible solution for LP2, which is $z_1 = b_1, z_2 = b_2, \ldots, z_k = b_k$ and $x_1 = x_2 = \cdots = x_n = 0$.

7. Prove that the following algorithm gives 2-approximation for minimum size vertex cover. That is, the set S output by the algorithm is a vertex cover and its size is at most twice of the optimal vertex cover.

$S \leftarrow$ empty set.

While the graph is non-empty

choose an edge (u, v) and put both its endpoints in S

delete u and v and all their incident edges from the graph

delete isolated vertices

Observe that the edges chosen during the algorithm form a matching in the given graph. Prove that this is an 1/2-approximation for maximum matching. That is, the matching obtained has size at least half of the maximum size matching.

8. Consider the following (approximation) algorithm for minimum size vertex cover. Construct examples to show that the approximation factor is not bounded by any constant, that is, the approximation factor increases with the input size.

$S \leftarrow$ empty set.

While the graph is non-empty

choose a vertex u with the highest degree and put it in S

delete u and all its incident edges from the graph

delete isolated vertices

---

9. **Maximum weight matching:** Given a graph with edge weights, the goal is to find a matching (set of disjoint edges) with maximum total weight. Write an integer linear program for the maximum weight matching problem. Now, remove the integer constraint, that is, variables are allowed to take any real value. We get a linear program. Find an example (a graph with edge weights), where the optimal value of the linear program is higher than the weight of the maximum weight matching. Interestingly, if the graph is bipartite then the two values are always equal.

10. Recall the greedy algorithm for minimum makespan problem. Prove that the analysis for 2-approximation we did was tight. That is, find an example where the makespan given by the greedy algorithm is roughly twice of the optimal makespan.

11. Consider a variant of the greedy algorithm, where we go over the job in decreasing order of processing times. Prove that this variant gives a 3/2-approximate solution.

    Hint: consider load on any processor. Split it into two parts: the last job assigned to that processor and the remaining jobs assigned to that processor. The total load of the remaining jobs is upper bounded by the average load per processor, because it must have been the minimum load among all processors at that time. Show that the last job has processing time at most 1/2 of the optimal makespan (assuming the processor was assigned at least two jobs).

12. Do you think your analysis above is tight? Do you see an example, where the solution obtained is indeed 3/2 times the optimal?