CS 6150: HW 6 – Optimization formulations, review

Submission date: Wednesday, December 6, 2023, 11:59 PM

This assignment has 5 questions, for a total of 50 plus 5 bonus points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Regression	5	
Sufficiency of constraints	16	
Minimum vertex cover revisited	16	
Optimal packaging	10	
Balancing sums	8	
Total:	55	

Question 1: Regression ..... **[5]**

The min-error linear regression problem is defined as follows: we are given n vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ in $\mathbb{R}^d$, along with real valued "outputs" $y_1, y_2, \ldots, y_n$. The goal in regression is to find an $x \in \mathbb{R}^d$ such that $\langle \mathbf{u}_i, x \rangle \approx y_i$ for all i. This is formalized as:

$$\text{find } x \in \mathbb{R}^d \text{ that minimizes } \max_{1 \le i \le n} |y_i - \langle \mathbf{u}_i, x \rangle| .$$

Phrase the min-error linear regression problem as a linear program.

Answer:

The issue with the provided function is that it uses two non-linear functions: max and absolute value. Because of this, the solution can't simply be $Minimize : \max_{1 \le i \le n} |y_i - \langle \mathbf{u}_i, x \rangle|$. We solve this by adding a new variable, W, and if we try to minimize W while ensuring it must be greater than or equal to each of the constraints, that will mimic the behavior of the max function. We can also duplicate each constraint and invert it to mimic the behavior of the absolute value function, since W now can't go too far in either the positive or the negative directions. The final LP is:

$$\begin{aligned} &\min W \text{ such that} \\ &W \ge y_i - \langle u_i, x \rangle \ \forall \ 1 \le i \le n \\ &-W \le y_i - \langle u_i, x \rangle \ \forall \ 1 \le i \le n \end{aligned}$$

Question 2: Sufficiency of constraints ..... **[16]**

Recall the maximum independent set problem: we are given an undirected graph $G = (V, E)$ and the goal is to find a subset S of vertices of the largest possible size such that no two vertices in S have an edge.

(a) **[6]** Consider the following optimization problem:

$$\begin{aligned} &\max \sum_{u \in V} x_u \text{ subject to} \\ &x_u^2 = x_u \text{ for all } u \in V \\ &x_u + x_v \le 1 \text{ for all } \{u, v\} \in E \end{aligned}$$

Prove that it captures the problem exactly; i.e., any feasible solution to formulation yields a feasible solution to original problem **and vice versa**.

Answer:

First, the constraint $x_u^2 = x_u$ is worthy of some scrutiny, since it contains a quadratic term. It can be rearranged to become $x_u(x_u - 1) = 0$. This equation can only be true if x_u is either 1 or 0, which means that every variable in this LP can only be 1 or 0.

Given a solution to MaxIndSet, S, we can assign every $x_u = 1$ for each $u \in S$, and 0 to all the others. This satisfies the first constraint, as every variable is either 1 or 0. For each $u, v \in E$ both u and v can't be chosen or otherwise it would fail the independent set requirement. Since only one or the other or neither can be chosen, this means it also satisfies the second constraint since the only way for $x_u + x_v > 1$ is if both are set to 1. Therefore, S also gives us a feasible solution to this LP.

Given a solution to the LP, L, we can add every u to our solution set where $x_u = 1$, and leave out all the others. Because of the first constraint, every x_u must be either 1 or 0, so anything with a non-zero value to it will be included in our solution set. Because of the second constraint, both u and v for all $u, v \in E$ can't both be selected, as otherwise $x_u + x_v > 1$. This means this solution set also satisfies the the independent set requirement, which means L gives us a feasible solution to MaxIndSet.

(b) [**5**] Now consider the relaxation in which the constraint $x_u^2 = x_u$ is replaced with $0 \leq x_u \leq 1$, for all u. This is now a linear programming relaxation. Consider the case in which the graph G is a clique with n vertices. Prove that the relaxation has a feasible solution $\{x_u\}$ such that $\sum_u x_u = n/2$. What is the value of the "true" maximum independent set in this case?

Answer:

If we assign $x_u = 1/2$ to everything, then our constraints are still met: $0 \leq 1/2 \leq 1$, and for any pair of vertices $1/2 + 1/2 \leq 1$. This gives us a solution value of $\sum_{u \in V} 1/2$, which is just $n/2$ as there are n vertices in V.

The real answer of course, is 1, since every vertex is connected to every other vertex so selecting any of them for the independent set disqualifies the rest.

(c) [**5**] Why are these answers different? Could we just use the formulation in part a) to solve Maximum Independent Set quickly?

Answer:

As discussed in part a), the first constraint is imposing the requirement that every variable only take integer values of 1 or 0. This is a type of linear program called an Integer Linear Program, or ILP. In relaxing this constraint, we have gone from an ILP to a regular LP, so the types of feasible solutions has expanded.

No, we cannot use the ILP to solve MaxIndSet quickly, as ILPs are NP-hard. It's no easier than MaxIndSet itself.

Question 3: Minimum vertex cover revisited ..... [**16**]

Recall the minimum vertex cover problem that we saw in HW 3. We are given an undirected graph $G = (V, E)$ and the goal is to select a subset S of the vertices of the smallest possible size that can "monitor" all the edges, i.e., for every edge $\{i, j\} \in E$, at least one of i, j is in S (so the objective is to minimize $|S|$ subject to the above).

(a) [**5**] What is a natural "decision version" of the problem? (Recall how we did this for independent set in class.)

Answer:

"Given a number k and graph G, is there a vertex cover for G of size $\leq k$?"

(b) [**5**] Show that the min vertex cover problem is in the class NP. Specifically, give a *verification algorithm* and describe the appropriate *certificate*.

Answer:

Our certificate will simply be the set of vertices that make up the vertex cover. To verify this solution, we can iterate through every edge in G and check if either endpoint is in the set. If we iterate through with a linear sweep as shown below, the runtime would be $O(nm)$, where n is the number of vertices and m is the number of edges.

Algorithm 1 Min Vertex Cover Verification

```
1:  for x in E do
2:      covered = false
3:      for y in S do
4:          if x.left = y or x.right = y then
5:              covered = true
6:          end if
7:      end for
8:      if covered = false then
9:          return false
10:     end if
11: end for
12: return true
```

(c) [**6**] We studied the linear programming (LP) relaxation for vertex cover. Recall that it is as follows:

$$\begin{aligned} &\text{minimize } \sum_{u \in V} x_u \text{ subject to} \\ &0 \leq x_u \leq 1 \text{ for all } u \in V \\ &x_u + x_v \geq 1 \text{ for all } \{u, v\} \in E. \end{aligned}$$

In class, we saw a rounding algorithm that takes a feasible solution $\{x_u\}$ to the LP above and produces a **feasible, binary** solution whose objective value is at most $2 \sum_u x_u$. Now, suppose we were lucky and the LP solution had all the x_u satisfying $x_u \in [0, 0.2) \cup (0.8, 1]$. In this case, prove that rounding produces a feasible, binary solution whose cost is at most $(1.25) \sum_u x_u$.

Answer:

For the same reasons than the normal rounding algorithm produces a feasible, binary solution, this also produces a feasible, binary solution since it's still the same algorithm. The only thing that's changed is the circumstances it's being used in, so we only need to demonstrate here the tighter approximation bound.

Let's consider the rounded variables, which we'll call y_u. Due to the way the values are split it's easy to see $y_u = 1$ when $x_u > 0.8$ and $y_u = 0$ when $x_u < 0.2$. We can now claim that $0.8y_u \leq x_u$ for all u. If $y_u = 0$, then it's trivial to see it holds in this case. if $y_u = 1$, Then we know it still holds since $0.8 \leq x_u$ when $0.8 < x_u \leq 1$, which it would need to be in order for $y_u = 1$. With this claim proved, we can simplify these summations to complete the proof:

$$\begin{aligned} 0.8 \sum_{u \in V} y_u &\leq \sum_{u \in V} x_u \\ \sum_{u \in V} y_u &\leq \frac{1}{0.8} \sum_{u \in V} x_u \\ &\leq 1.25 \sum_{u \in V} x_u \end{aligned}$$

Question 4: Optimal packaging ..... **[10]**

With the holiday season around the corner, company Bezo wants to minimize the number of shipping boxes. Let us consider the one dimensional version of the problem: suppose a customer orders n items of lengths $a_1, a_2, \ldots, a_n$ respectively, and suppose $0 < a_i \leq 1$. The goal is to place them into boxes of length 1 such that the total **number of boxes** is minimized.

It turns out that this is a rather difficult problem. But now, suppose that we have only a small number of *distinct lengths*. I.e., suppose that there is some set $L = \{s_1, \ldots, s_r\}$ such that all the $a_i \in L$ (and think of r is as a small constant). Devise an algorithm that runs in time $O(n^{3r})$ (or better), and computes the optimal number of boxes.

[*Hint:* first find all the possible "configurations" that can fit in a single box. Then use dynamic programming.]

Answer:

Given the set of elements $a_1, a_2, \ldots, a_n$, we can derive the set of unique sizes, $L = \{s_1, \ldots, s_r\}$, and we can come up with a set,

$$R = \{c_1, \ldots, c_r\} \text{ where } c_i = |\{a_i | \text{length of } a_i = s_i\}| \tag{1}$$

Note that $0 \leq c_i \leq n$. Now consider a box of size 1. Note that we can write the content of the box in terms of the number of elements from each type, i.e., the contents of the box $B_i = \{x_1, x_2 \ldots, x_r\}$ where $0 \leq x_i \leq c_i$ and $\sum_{i=1}^{r} x_i \leq 1$. Since $c_i \leq n$, we could have at most $O(n)$ values for each x_i and therefore,

we can have at most $O(n^r)$ different combinations. Let the set of this combinations be S. We can now try to write the recurrence of the dynamic program. Let $f(R)$ be the number of boxes needed with the R configuration given the S set of combinations. Before we define the DP, we first define a couple of new operations that allows us to compare two different configurations. We first define an operation $A - B = \{a_1 - b_1, a_2 - b_2, \ldots, a_r - b_r\}$ (where $A = \{a_1, \ldots, a_r\}, B = \{b_1, \ldots, b_r\} \in S$) which allows us to capture the number of remaining items of each size if we started with configuration A and removed items of configuration B from it. We also define a comparison operator $\geqslant$ that lets us check if given two configurations A, B, we can remove elements of B from A such that we do not have a case where B has more elements than A for some item size. We can formally define this as $A \geqslant B \iff \min_{i=1}^{r} a_i - b_i \geq 0$. Then,

$$f(R) = 1 + \min_{\hat{S} \in S, R \geqslant \hat{S}} f(R - \hat{S})$$

The recurrence takes a configuration R as the input and picks a configuration $\hat{S}$ such that using 1 box to cover that configuration and covering the remaining recursively to find the minimal number of boxes needed. Note that we only consider configurations such that $R \geqslant \hat{S}$ because each time we only need to consider configurations $\hat{S}$ such that the required number of elements of each length in $\hat{S}$ does not exceed the available number of items. We can give a DP algorithm for this,

Algorithm 2 Optimal Packing

Input: A set of elements $\{a_1, a_2, \ldots, a_n\}$

1: Generate $R = \{c_1, c_2, \ldots, c_r\}$ as given by equation 1
2: Generate S (the possible combinations)
3: OptimalPacking$(R) \to \infty$
4: **for** $\hat{S} \in S$ such that $R \geqslant \hat{S}$ **do**
5: Compute/Lookup OptimalPacking$(R - \hat{S})$
6: Set OptimalPacking$(R) \to \min\{$OptimalPacking$(R), 1 +$ OptimalPacking$(R - \hat{S})\}$
7: **end for**
8: **return** OptimalPacking(R)

Correctness: We can see that our algorithm follows the recurrence and therefore it goes through all possible cases for

Complexity: Generating R takes at most $O(nr)$ time. Note that the set S is of size $O(n^r)$ so we can construct it in a brute force manner in $O(n^r)$ time. Note that since each problem takes $O(n^r)$ time to go over all the possible configurations, we get the total time complexity would be $O(n^{2r})$.

Question 5: Balancing sums ..... **[8]**

Suppose we have n real numbers $a_1, a_2, \ldots, a_n \in (0, 1)$. The "balancing" problem asks to find $\pm$ *signs* such that the signed sum of the a_i is as small as possible. Formally, the goal is to find signs $s_i \in \{+1, -1\}$ for every $1 \leq i \leq n$ such that $|\sum_i s_i a_i|$ is minimized. Suppose for a moment that a_i are all rational numbers (fractions) with a common denominator p. Give an algorithm for this problem with running time polynomial in p, n.

Answer:

Since $0 < a_i < 1$ and since a_i is in the form $a_i = q_i/p$, we can see that $q_i = pa_i$ are integers such that $0 < q_i < p$. Now we can see that the problem of finding s_i signs such that $|\sum_i s_i a_i|$ is minimized is equivalent to finding s_i signs such that $|p\sum_i s_i a_i| = |\sum_i s_i q_i|$. Since q_i are integers such that $q_i < p$, the absolute sum can only take values from $0 < |\sum s_i q_i| < np$.

Consider the set of elements $Q = \{q_1, q_2, \ldots, q_n\}$ where $q_i = a_i p$. Let,

$$\begin{aligned} M(s_1, s_2, \ldots, s_n) &= \sum_i s_i q_i \\ &= \sum_{i;s_i=1} s_i q_i + \sum_{i;s_i=-1} s_i q_i \\ &= \sum_i q_i - 2 \sum_{i;s_i=-1} q_i \end{aligned}$$

be the sum of values given a set of signs, $s_1, s_2, \ldots, s_n$.Note that $\sum_{i;s_i=-1} q_i$ can take values from $\{1, 2, \ldots, np-1\}$. Now all we need to figure out is if we can use a subset of elements from $\{q_1, q_2, \ldots, q_n\}$ to get a value closer to $\frac{\sum_i q_i}{2}$.

We can define the function $f(i, R)$ be 1 if and only if we can make R using only the elements from 1 to i (and 0 otherwise). We can see that,

$$f[i][R] = \begin{cases} f[i-1][R] & q_i > R \\ \max\{f[i-1][R], f[i-1][R-q_i]\} & q_i \leq R \end{cases}$$

and $f[i][0] = 1$. Note that this is true since in order to make R using a subset from $\{q_1, q_2, \ldots, q_i\}$, we either need to be able to make $R - q_i$ (this captures the notion of using q_i in the sum when $R \geq q_i$) or R (not using q_i) using a subset from $\{q_1, q_2, \ldots, q_{i-1}\}$. If either one is 1, we get $f[i][R] = 1$ and 0 otherwise (acts as an OR gate).

Algorithm 3 Minimize the mangitude of signed sum

Input: The real numbers $a_1, a_2, \ldots, a_n$ and denominator p

1: Let $q_i = a_i p$ for all i
2: Let $S = \sum_{i=1}^{n} q_i$
3: Define $f[1 \ldots n][0 \ldots (np-1)]$ and initiate $f[i][0] = 1$ for all i
4: **for** $i \in \{1, 2, \ldots, n\}$ **do**
5: **for** $R \in \{1, 2, \ldots, np-1\}$ **do**
6: Compute $f[i][R] = \begin{cases} f[i-1][R] & q_i > R \\ \max\{f[i-1][R], f[i-1][R-q_i]\} & q_i \leq R \end{cases}$
7: **end for**
8: **end for**
9: Let $M \to \infty$
10: **for** $R \in \{0, 1, 2, \ldots, np-1\}$ **do**
11: **if** $f[n][R] = 1$ **then**
12: $M \to \min\{M, |S - 2R|\}$
13: **end if**
14: **end for**
15: **return** M

Correctness: Note that given our definition of f, $f[n][R]$ gives whether we can make R using a subset of $\{q_1, q_2, \ldots, q_n\}$ for all $R \in \{0, 1, 2, \ldots, np-1\}$. Then since we know the minimum absolute sum is the minimum of $|S - 2R|$ for the possible R values from $R \in \{0, 1, 2, \ldots, np-1\}$. Since we calculate all possible R values, iterating through all possible values and picking the minimum gives us the desired result.

Complexity: Note that for this program, we need $O(n^2 p)$ space and computing each value takes constant time so the overall time complexity is $O(n^2 p)$. We can see that lines 1,2,3 in the algorithm takes $O(n)$ time. And the for loop on line 10 takes $O(np)$ time. Therefore, overall time complexity is $O(n^2 p)$ which is polynomial in n, p.