CS 6150: HW 6 –Payman Behnam (u1078370)
Optimization formulations, review

Submission date: Thursday, December 5, 2019, 11:59 PM

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

Question	Points	Score
Regression	5	
Sufficiency of constraints	10	
Minimum vertex cover revisited	13	
Optimal packaging	10	
Balancing sums	12	
Total:	50	

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

The min-error linear regression problem is defined as follows: we are given n vectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$ in $\mathbb{R}^d$, along with real valued "outputs" $y_1, y_2, \dots, y_n$. 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.

Solution We know that

$$minimiztion \max_{1 \le i \le n} |y_i - \langle \mathbf{u}_i, x\rangle| .$$

can be represented as

$$minimize R = \max_{1 \le i \le n} |y_i - \langle \mathbf{u}_i, x\rangle| .$$

Suppose we have N points. Then, based on the definition of absolute value we can have the following:

$$R - (y_1 - \langle \mathbf{u}_1, x\rangle) \ge 0 \Rightarrow -R - \langle \mathbf{u}_1, x\rangle \le -y_1$$

$$R + (y_1 - \langle \mathbf{u}_1, x\rangle) \ge 0 \Rightarrow -R + \langle \mathbf{u}_1, x\rangle \le y_1$$

$$R - (y_2 - \langle \mathbf{u}_2, x\rangle) \ge 0 \Rightarrow -R - \langle \mathbf{u}_2, x\rangle \le -y_2$$

$$R + (y_2 - \langle \mathbf{u}_2, x\rangle) \ge 0 \Rightarrow -R + \langle \mathbf{u}_2 x\rangle \le y_2$$

$$.$$

$$.$$

$$.$$

$$R - (y_{n-1} - \langle \mathbf{u}_{n-1}, x\rangle) \ge 0 \Rightarrow -R - \langle \mathbf{u}_{n-1}, x\rangle \le -y_{n-1}$$

$$R + (y_{n-1} - \langle \mathbf{u}_{n-1}, x\rangle) \ge 0 \Rightarrow -R + \langle \mathbf{u}_{n-1}, x\rangle \le +y_{n-1}$$

$$R - (y_n - \langle \mathbf{u}_n, x\rangle) \ge 0 \Rightarrow -R - \langle \mathbf{u}_n, x\rangle \le -y_n$$

$$R - (y_n - \langle \mathbf{u}_n, x\rangle) \ge 0 \Rightarrow -R + \langle \mathbf{u}_n, x\rangle \le +y_n$$

This can be represented also as :

$$-R - \sum_i \langle \mathbf{u}_i, x\rangle \le -y_i$$

$$-R + \sum_i \langle \mathbf{u}_i, x\rangle \le y_i$$

Suppose that somehow you can represent $< u_i, x >$ as a multiplication of u_i and x; We have:

$$\begin{bmatrix} -1 & -u_1 \\ -1 & +u_1 \\ \vdots & \vdots \\ -1 & -u_n \\ -1 & u_n \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} \le \begin{bmatrix} -y_1 & \\ y_1 & \\ -y_2 & \\ y_2 & \\ \vdots & \vdots \\ -y_n & \\ y_n & \end{bmatrix} \tag{1}$$

Which is nothing except $Av \le B$.

$$A = \begin{bmatrix} -1 & -u_1 \\ -1 & +u_1 \\ \vdots & \vdots \\ -1 & -u_n \\ -1 & u_n \end{bmatrix} \tag{2}$$

$$v = \begin{bmatrix} r \\ x \end{bmatrix} \tag{3}$$

$$B = \begin{bmatrix} -y_1 & \\ y_1 & \\ -y_2 & \\ y_2 & \\ \vdots & \vdots \\ -y_n & \\ y_n & \end{bmatrix} \tag{4}$$

If we define f like the following: $f = [0\ 1]$

We have :

Objective: minimize max of fx Subject to $Av \leq B$

Reference: https://math.stackexchange.com/questions/47944/linear-regression-for-minimizing-the-maximum-of-the-residuals

Question 2: Sufficiency of constraints .... **[10]**

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) [**5**] Consider the following optimization problem:

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

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

Solution

Let's see how it is formed. Based on $x_u^2 = x_u$ constraint, we know that $x_u \in \{0, 1\}$. S can be defined as S $= \{u | u \in V, x_u = 1\}$. Due to the constraint that $x_u + x_v \leq 1$ for all $u, v \in E$, we can have the following options: $(x_u, x_v) = (0, 0)$ or $(x_u, x_v) = (0, 1)$ or $(x_u, x_v) = (1, 0)$. There is no solution like $(x_u, x_v) = (1, 1)$, which means that we cannot select both x_u and x_v if $u, v \notin E$. This means that any feasible solution to formulation yields a feasible solution to original problem.

Now, we need to prove that a feasible solution to the original problem yields a feasible solution to formula. For maximum independent set, we should choose some vertex and dismiss others. We need a binary variable that define whether the vertex is chosen or not. That would be x_u and we can make it binary using the $x_u^2 = x_u$ constraint. Now, we have to say that for each pair the chosen vertcis, there is no edge between them, which happens through the constraint $x_u + x_v \leq 1$ for all $\{u, v\} \in E$.

Due to this case, the (1,1) is not acceptable which means that if we have $u, v \in E$, only one can be included in the maximum independent set. The task would be finding the maximum value for the set, which happens through the object function: $max \sum_{u \in V} x_u$

(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?

Based on the relaxation, lets suppose that $x_u = 0.5$. $0 \leq x_u \leq 1$ and if $x_v = 0.5$ then $x_u + x_v \leq 1$ for all $u, v \in E$. Hence, we meet the constraint.

We are looking for max $\sum_{u \in V} x_u$. Clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. Since we have n vertices, we would have $\sum_u x_u = n \times 0.5 = \frac{n}{2}$. In a clique, every two vertices are adjacent. So, only one vertex can be chosen as a "true" maximum independent set.

Question 3: Minimum vertex cover revisited .... [**13**]

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

Solution

A natural "decision version" of the problem would be:

Does the $G = (V, E)$ has a a subset S of the vertices with size K that we put camera on them and "monitor" all the edges?

To solve optimization using decision, first, we should find minimum k. Start with k=1 and check if only one vertex is enough. If not, then we increase k until reach a point that that k vertex can cover all the edges. For the set, we pick any vertex u and continue with removing all the edges connects to the G (G'). If G' is empty we are done, otherwise we continue the same procedure with G'.

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

Solution

By the definition, if a problem is in NP, then, a 'certificate' will give an instance to verifier to verify it in polynomial time. If instance I is a YES instance, there exists a witness w such that verifier outputs YES.

If instance I is a NO instance, for any witness w, verifier outputs NO.

The certificate for the minimum vertex cover problem is a subset V' of V. We can check if the set V' is a vertex cover of size k using the following verifier :

Algorithm 1 Minimum vertex cove verifier

```
1: procedure MINIMUM VERTEX COVE VERIFIER
2:     int count = 0                                  ▷ O(1)
3:     For (each vertex v in V')                      ▷ O(n)
4:     remove all edges adjacent to v from E;
5:     count ++ ;
6:     if count = k and E is empty then
7:         the given solution is correct;             ▷ O(1)
8:     else
9:         the given solution is incorrect;           ▷ O(1)
```

The complexity would be O(N).

Reference: https://www.geeksforgeeks.org/proof-that-vertex-cover-is-np-complete/

(c) [**5**] We studied the linear programming (LP) relaxation for vertex cover. Recall that it is as follows:

$$\text{minimize } \sum_{u \in V} x_u \text{ subject to}$$
$$0 \le x_u \le 1 \text{ for all } u \in V$$
$$x_u + x_v \ge 1 \text{ for all } \{u, v\} \in E.$$

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 $(1.25)\sum_u x_u$.

Solution: The input of rounding algorithm is x_u and the output would be y_u as

$$y_u = \begin{cases} 0, & \text{if } x_u \in [0, 0.2) \\ 1, & \text{if } x_u \in (0.8, 1] \end{cases} \tag{5}$$

Due to the condition that $x_u + x_v \ge 1$, at least one of the u or v must be in (0.8 , 1]. Suppose $x_u \in (0.8, 1]$, then $x_v \in (0.8, 1]$ or $x_v \in [0, 0.2)$ in a way that still $x_u + x_v \ge 1$. Lets consider worst case when $x_u \in (0.8, 1]$ then one possible value for x_v would be in (0.8 , 1) again. Things like 0.80000001+0.80000001 ; based on the rounding for y_u, both x_u and x_v are rounded to 1. So, for the original problem the $x_u + x_v > 1.6$ while in the rounded one $y_u + y_v \ge 2$. Hence rounding produces a pair whose cost is $\frac{2}{1.6} = 1.25\times$ bigger than original solution. Since, this should happen for all the selected pairs, by adding summation we would have $\sum_u y_u = 1.25 \sum_u x_u$. Since we consider worst case (for other possibility the ration would be less considering the fact that the sum of original can be a little bit grater than 1 while for the rounded one it would be 1), we claim that rounding produces a feasible, binary solution whose cost is at most $(1.25)\sum_u x_u$.

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 $0 < a_i \le 1$. 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 $O(n^r)$, 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.]

We have n elements and r sizes. $A = \{a_1, a_2, ..., a_n\}$ and $L = \{s_1, \ldots, s_r\}$. Let define B = $\{y_1, y_2, ..., y_r\}$ such that y_i is the number of elements with size size s_i. C is set of all the possible configurations that is fit into a box.

$$C = \{(x_1, x_2, ..., x_r) | 0 \le x_i \le y_i \le n \text{ for all i=1,...,r} and \sum_i xisi \le 1\}$$

.

It means that we have to choose some elements (x_i) of size s_i such that the total size doesn't exceed the maximum size of the box and we have to do it such that all elements are fit in minimum number of boxes. The number of configuration of C is (n^r). Since we have to do it for all n elements and each one takes (n^r) the complexity would be O(n^{2r}). This gives an intuition for solving the problem. To come up with a recursion and solve the problem, we can see that this is like the vectorized version of the coin change problem with following distinctions:

We dont have infinite number of types of each coin (i.e., here the number of elements with size s_i is y_i and total number of elements is not infinity. So it can be solved using DP.

Suppose that:

$D[j_1]...[j_r]$ are minimal number of boxes needed with some elements of specific type .

If we know all possible combinations of elements into a single box, then the optimal solution to $D[j_1]...[j_r]$ is obtained from the optimal solution of $D[j_1 - a_1]...[j_r - a_r]$ by considering all possible elements tuples $(x_1, ..., x_r)$ such that $x_1 s_1 + ... + x_r s_r \leq 1$ fit into one box. So an optimal solution is obtained by stepping forward with one box from some optimal solution with one less box which is already optimized.

The set C can be computed using the following algorithm.

Algorithm 2 ComputeCD (L,B,r,n)

```
1: procedure ComputeCD(L,B,r,n)
2:     C = $\emptyset$
3:     For ($x_1 = 0$ to $y_1$)
4:       For ($x_2$ =0 to $y_2$)
5:     .....
6:         For ($x_r$ =0 to $y_r$)
7:     if ($x_1 s_1 + ... + x_r s_r \leq 1$) then
8:          C = C U $(x_1, x_2, , .., x_r)$;
9:     $D[a_1]...[a_r] \leftarrow 1$
10:    return(C,D);
```

We need to do n work in r nested loops. The complexity of the above algorithm would be O(n^r),

The minimum number of boxes is computed using the following procedure.

Algorithm 3 MinBox (L,B,r,n)

```
1: procedure MinBox(L,B,r,n)
2:     (C,D) $\leftarrow$ ComputeCD(L,B,r,n)
3:     D[0]...[0] $\leftarrow$ 0
4:     For ($j_1 = 0$ to $y_1$)
5:       For ($j_2$ =0 to $y_2$)
6:     .....
7:         For ($j_r$ =0 to $y_r$)
8:     $D[j_1]...[j_r] \leftarrow \min_{(x_1,...,x_r) \in C} \{1 + D[j_1 - x_1]...[j_k - x_k] : j_i \geq x_i, i = 1, ..., r\}$
9:     return(D $[y_1]...[y - r]$)
```

For each of the at most O(n^r) sub-problems, we have to compute the min over all configurations by doing O(n^r) work. Hence, the complexity would be O(n^{2r}) while the space complexity (requirement) is O(n^r).

Proof of correctness: Please refer to the above explanation about how I computed B and C and come up with the recursion. we should transition from smaller to larger problem instances. We cannot remove an element and ask what the optimal solution is to the sub-problem since the number of required boxes can increase or remain the same. The proposed mapping mechanism map n element to r type of elements such that we can have a summation of $x_i s_i$ less than 1 for each box.

Reference: Problem 5 in https://amininima.wordpress.com/2013/08/01/dynamic-programming/

Question 5: Balancing sums .... **[12]**

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 $1 \le i \le n$ such that $|\sum_i s_i a_i|$ is minimized.

(a) **[6]** 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.

Solution Based on the problem , the that a_i are all rational numbers (fractions) with a common denominator p. So, we have $a_i = \frac{q_i}{p}$ and we know that $1 \le q_i < p$. We have p possible values for a_i. We would have $sum = \sum_i s_i \frac{q_i}{p}$. Since we have n numbers, $-n{<}sum{<}n$. Since we have n numbers and each numbers can have p possible values, we can have np values.

We can solve the problem using dynamic programming by inspiring standard 0/1 knapsack problem. We have 2 options: either flip the sign of element or not filliping that one (it is like to take an item or not in the knapsack). The absolute value of sum of all elements of the array without flipping would be like the weight in knapsack problem.

If we solve the problem such that find minimum number of required filliping of signs, then we would have the favorite of the question.

Optimal Substructure: We put elements in the array. Let dp[i][j] be the minimum number of flips required in the first i elements of the array to make the sum equal to j.

If sign of ar[i − 1] is not flipped to make sum = j, dp[i][j] = dp[i − 1][j − A[i − 1]]

If sign of ar[i − 1] is flipped to make sum = j, dp[i][j] = min(dp[i][j], dp[i − 1][j + A[i − 1]])

Algorithm 4 Balancing Sum

```
procedure BALANCING SUM(int A[], int n)
    unordered-map dp[2]                          ▷ Array of unordered-map of size=2
    bool flag = 1;                ▷ boolean variable used for toggling between maps
    sum = 0;
    For ( i = 0; i <n; i++) {sum += A[i];}
    For ( i = -sum; i ≤ sum ; i++) {dp[0][i] = INT-MAX;}
    dp[0][0] = 0;
    For ( i = 0; i <n; i++) {
            For ( i = -sum; i ≤ sum ; i++) {
             dp[flag][j] = INT-MAX;;
    if ($j - A[i-1] \le sum \&\& j - A[i-1] \ge -sum$) then
        dp[flag][j] = dp[flag xor 1][j - A[i - 1]];
    if (j + A[i - 1] ≤ sum && j + A[i - 1] ≥ -sum && dp[flag xor 1][j + A[i - 1]] ≠ INT-MAX) then
        $dp[flag][j] = min(dp[flag][j], dp[flagxor1][j + A[i-1]] + 1);$
            }
    flag = flag xor 1
    }
    For ( i = 0; i ≤ sum ; i++) {
    if ($dp[flagxor1][i] \ne INT - MAX$) then{
        return dp[flag xor 1][i];
    }
    return n - 1;
```

Proof of correctness: Please pay attention to what I said in the above for how I come up with the algorithm and recursion. We can use DP since there are overlapping subproblems. We store results of subproblems in a table and reuse them. Please note because the sum of elements can be negative we do not use a direct 2D array because in dp[i][j], j is the sum and indexes of the array cannot be negative. Instead we we use an array of Hash maps. Size of the array will be n + 1. In worst case we will flip n-1 elements.

Complexity: The complexity would be O($n \times sum$). For the sum we would have $sum = |\sum_i a_i| = \sum_i \frac{q_i}{p} \leq n$. The largest possible value of the sum would be when all numbers are the max possible values which is $\frac{p-1}{p}$. Then the sum would be $n \times \frac{p-1}{p}$. The time complexity would be O($\frac{n^2p-n^2}{p}$) = O($n^2 - \frac{n^2}{p}$). If p is big enough the complexity would be O(n^2)

(b) **[6]** Suppose we choose the signs uniformly at random. Prove that the sum $\sum_i s_i a_i$ has magnitude $\leq 4\sqrt{n}$ with probability at least 3/4. [*Hint:* recall the Brownian motion problem from HW5!]

Let $X = \sum_i s_i a_i$ be a random variable. Signs are chosen uniformly at random so we have $E[s_i] = -1 \times Pr[s_i = -1] + 1 \times Pr[s_i = 1] = \frac{-1}{2} + \frac{1}{2} = 0$ So based on the linearity of expectation, E[X] = E[$\sum_i s_i a_i$] = $\sum_i E[s_i]a_i = 0$. Var[s_i] = E[s_i^2] - E[s_i]^2= $\frac{1}{2} + \frac{1}{2} - 0 = 1$ For the variance we have Var[X] = Var[$\sum_i s_i a_i$] = $\sum_i Var[s_i a_i] = a_i^2 \sum_i Var[s_i]$. Since $0 < a_i < 1$, then $0 < a_i^2 < 1$ and Var[s_i] =1;Hence, $Var[X] < n$ Based on the Chebyshev's inequality, we have:

$$\Pr(|X - E[X]| \geq k \times \sqrt(Var[X])) \leq \frac{1}{k^2}$$

Pr($|X - 0| \geq k \times \sqrt{n}$) $\leq \frac{1}{k^2}$ suppose that k =4:

Then we would have : Pr($|X - 0| \leq 4 \times \sqrt{n}$) $= 1 -$ [Pr($|X - 0| \geq 4 \times \sqrt{n}$)] = 1 - $\frac{1}{4^2} = \frac{15}{16} > \frac{3}{4}$