Collaborators

Ben Nelson, Corbin Baldwin and I collaborated for this assignment.

Question 1: Basics of linear programming ..... **[9]**

Consider the feasible set of the following set of inequalities in two variables x, y:

$$\begin{aligned} x + y &\geq 1, \\ x + 2y &\leq 4, \\ y &\leq 2. \end{aligned}$$

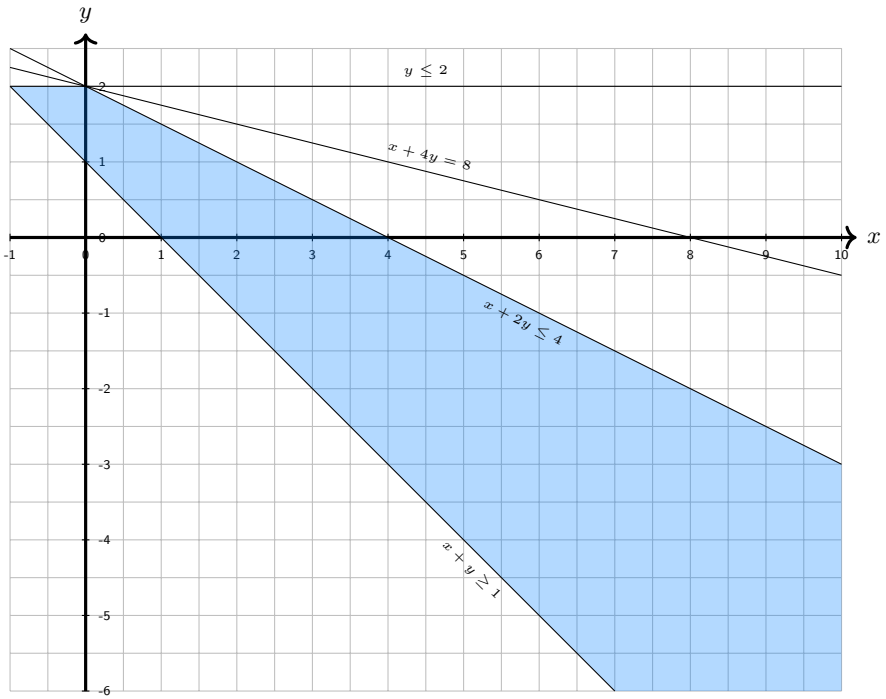
(a) **[3]** Draw the feasible region on the plane (you may also draw on paper, roughly to scale, and attach a scanned image).

(b) **[2]** What is the maximum value of $x + 4y$ subject to (x, y) being in the feasible region?

(c) **[2]** Answer the same for $x + y$.

(d) **[2]** Find a point (x, y) that is the intersection of two of the lines defining the equations above, but is not a corner point of the feasible region.

(a) The feasible region is shown below in cyan. It is unbounded on the bottom-right.



(b) The maximum value of $x + 4y$ subject to (x, y) being in the feasible region will be at the corner $(0, 2)$ as shown above.

(c) We can try and find the maximum value of $x + y$ subject to (x, y) being in the feasible region, by moving the line $x + y = 1$ (the line passing through $(0, 1)$ and $(1, 0)$) to the right and keeping it parallel to the original line. Since the feasible region is unbounded at the bottom right, the maximum value cannot be found as it will be infinite.

(d) The lines $x + y = 1$ and $x + 2y = 4$ intersect at $(-2, 3)$, which as can be seen from the figure above, is not a corner point of the feasible region.

Question 2: Paths and spanning trees as linear optimization ..... **[11]**

In class, we saw the following idea to write the shortest path problem as linear optimization over a $\{0, 1\}$

(discrete) domain. Suppose we have an undirected graph $G = (V, E)$, and suppose that w_e denotes the weight (or length of edge e). The goal is to find the shortest total length path between s and t. The idea was to choose binary variables x_{ij} that now correspond to *directed edges* (going from $i \to j$), and the idea is that we would set $x_{ij} = 1$ if the shortest path from s to t uses the directed edge (i, j).

(a) [**4**] Formally write down the objective and the set of constraints (discussed roughly in class and in the hint below). Note that you need to argue that any feasible solution to the constraints you write down yields a valid path in the graph.

[*Hint:* every vertex other than s, t must have precisely one edge coming in and going out.]

(b) [**7**] Now consider the minimum spanning tree problem, in which we have an undirected graph with weights on the edges, and the goal is to find the connected subgraph (i.e., one with a path between every pair of vertices in V) that minimizes the total weight of the selected edges.

(a) For each edge $\{i, j\}$ in the graph, where i and j are vertices, introduce two indicator variables x_{ij} and x_{ji} both $\in \{0, 1\}$ where $x_{ij} = 1$ if the edge $\{i, j\}$ in the direction $i \to j$ is on the shortest path between s and t and similarly $x_{ji} = 1$ if the edge $\{j, i\}$ in the direction $j \to i$ is on the shortest path between s and t. Let $\Gamma(i)$ denote the set of neighboring vertices of vertex i.

$$\sum_{i \in \Gamma(s)} x_{si} = 1 \tag{1}$$

$$\sum_{i \in \Gamma(s)} x_{is} = 0 \tag{2}$$

$$\sum_{i \in \Gamma(t)} x_{ti} = 0 \tag{3}$$

$$\sum_{i \in \Gamma(t)} x_{it} = 1 \tag{4}$$

Equation (1) is the constraint that exactly one edge leaving vertex s should be on the shortest path between s and t. Equation (2) is the constraint that no edge connected to vertex s is on a path entering s. Similarly equation (4) is the constraint that exactly one edge entering vertex t should be on the shortest path between s and t and equation (3) is the constraint that no edge connected to vertex t is on a path leving t.

$$\forall i \notin \{s, t\} \sum_{j \in \Gamma(i)} x_{ji} = \sum_{j \in \Gamma(i)} x_{ij} \tag{5}$$

Equation (5) is the constraint that the number of paths entering an internal node is equal to the number of paths leaving it. Subject to the above constraints, we need to minimize the length of a path. That is done by minimizing the sum of the edge weights on the path $\sum_{ij} w_{ij} x_{ij}$ where i and j are any two neighboring vertices.

Due to constraint (1), a path is guaranteed to start from vertex s and constraint (5) ensures that there is an unbroken chain of edges starting from the vertex that s is connected to. Finally constraint (4) ensures that this unbroken chain of edges ends at vertex t. This unbroken chain of edges from s to t is a path from s to t and minimizing the objective $\sum_{ij} w_{ij} x_{ij}$ ensures that this is the shortest path between the two vertices.

So the linear program is

$$\begin{aligned}
\forall \text{ vertices } i,j \text{ minimize } & \sum_{ij} w_{ij}x_{ij} \text{ subject to} \\
& \sum_{i\in\Gamma(s)} x_{si} = 1 \\
& \sum_{i\in\Gamma(s)} x_{is} = 0 \\
& \sum_{i\in\Gamma(t)} x_{ti} = 0 \\
& \sum_{i\in\Gamma(t)} x_{it} = 1 \\
\forall i \notin \{s,t\} & \sum_{j\in\Gamma(i)} x_{ji} = \sum_{j\in\Gamma(i)} x_{ij}
\end{aligned}$$

(b) A minimum spanning can be thought to be comprised of edges selected from the original graph such that the shortest distance between any two distinct nodes is a subset of these selected edges. Similar to the above case of shortest path, we can have the constraints to be

$$\forall \text{ distinct vertices } i,j \sum_{u\in\Gamma(i)} x_{iu} = 1 \tag{6}$$

$$\sum_{u\in\Gamma(i)} x_{ui} = 0 \tag{7}$$

$$\sum_{u\in\Gamma(j)} x_{ju} = 0 \tag{8}$$

$$\sum_{u\in\Gamma(j)} x_{uj} = 1 \tag{9}$$

$$\forall u \notin \{i,j\} \sum_{v\in\Gamma(u)} x_{vu} = \sum_{v\in\Gamma(u)} x_{uv} \tag{10}$$

The objective function to be minimized will be

$$\forall \text{ distinct vertices } i,j \sum_{uv} w_{uv}x_{uv}, \text{ where } u \text{ and } v \text{ are neighboring vertices} \tag{11}$$

Constraints (6) through (10) will ensure that there is a path between each distinct pair of nodes $\{i,j\}$ and minimizing the objective (11) will ensure that the path between i and j is the shortest one. The set of all edges so selected will form a minimum spanning tree since the path between any two distinct nodes is the shortest.

Question 3: Checking feasibility vs optimization ..... **[12]**

Some of the algorithms for linear programming (e.g. simplex) start off with one of the corner points of the feasible set. This turns out to be tricky in general. In this problem, we will see that in general, finding one feasible point is as difficult as actually performing the optimization!

Consider the following linear program (in n variables $x_1, \ldots, x_n$, represented by the vector x):

$$\begin{gathered}\text{minimize } c^T x \text{ subject to} \\ a_1^T x \geq b_1 \\ a_2^T x \geq b_2 \\ \cdots \\ a_m^T x \geq b_m.\end{gathered}$$

(a) **[6]** Suppose you know that the optimum value (i.e. the minimum of $c^T x$ over the feasible set) lies in the interval $[-M, M]$ for some real number M (this is typically possible in practice). Suppose also that you have an **oracle** that can take any linear program and say whether it is feasible or not. Prove that using $O(\log(M/\epsilon))$ calls to the oracle, one can determine the optimum value of the LP above up to an error of $\pm\epsilon$, for any given accuracy $\epsilon > 0$. [*Hint:* can you write a new LP that is feasible only if the LP above has optimum value $\leq z$, for some z?]

(b) **[6]** Part (a) gave a way to find the optimum *value*. Now suppose we wish to find the optimum solution (i.e., the best x). Suppose we knew that in the optimum solution, $x_i \in [-M, M]$ for all i. Show how to find each x_i to an error $\pm\epsilon$ using $O(n \log(M/\epsilon))$ calls to the oracle.

(a) Consider the following linear program which is the original LP plus an additional constraint:

$$\begin{gathered}\text{minimize } c^T x \text{ subject to} \\ c^T x \leq \alpha \\ a_1^T x \geq b_1 \\ a_2^T x \geq b_2 \\ \cdots \\ a_m^T x \geq b_m.\end{gathered}$$

Shown below is an algorithm for finding out the optimum value of the LP given in the question for any given accuracy $\epsilon > 0$, which is passed in as a parameter.

Algorithm 1 Find optimum value of LP

```
1: procedure LP_Opt_Value(ϵ)
2:     lower_limit = −M
3:     upper_limit = M
4:     α = (upper_limit − lower_limit)/2
5:     counter = 0
6:     while counter < log(M/ϵ) do
7:         oracle_answer = ASK_ORACLE_IF_FEASIBLE(α)
8:         if oracle_answer == true then
9:             upper_limit = α
10:        else
11:            lower_limit = α
12:        end if
13:        α = (upper_limit − lower_limit)/2
14:    end while
15:    return (lower_limit, upper_limit)
16: end procedure
17: procedure ASK_ORACLE_IF_FEASIBLE(α)
18:    Ask the oracle if LP: minimize c^T x subject to c^T x ≤ α, a_1^T x ≥ b_1, ... a_m^T x ≥ b_m is feasible
19:    if oracle says yes then
20:        return true
21:    else
22:        return false
23:    end if
24: end procedure
```

Each iteration of the while loop divides the range into half and asks the oracle if the new LP is feasible. If it is, then the middle of the range becomes the new upper end of the range, else it becomes the new lower end of the range. With each iteration, the range becomes half. So after $\log\left(\frac{M}{\epsilon}\right)$ iterations, the range becomes $\frac{2M}{2^{\log\left(\frac{M}{\epsilon}\right)}} = \frac{2M}{\frac{M}{\epsilon}} = 2\epsilon$. So after the iterations, the optimum value of the LP will be within $\pm\epsilon$, for any given accuracy $\epsilon > 0$.

(b) Consider the algorithm shown below on lines similar to the above algorithm.

Algorithm 2 Find optimum solution of LP

```
1: procedure LP_Opt_Solution(ϵ)
2:     solution_component = 1
3:     while solution_component ≤ n do
4:         lower_limit = −M
5:         upper_limit = M
6:         α = (upper_limit − lower_limit)/2
7:         counter = 0
8:         while counter < log(M/ϵ) do
9:             oracle_answer = ASK_ORACLE_IF_FEASIBLE(solution_component, α)
10:            if oracle_answer == true then
11:                upper_limit = α
12:            else
13:                lower_limit = α
14:            end if
15:            α = (upper_limit − lower_limit)/2
16:        end while
17:        x_solution_component = (lower_limit, upper_limit)
18:        solution_component = solution_component + 1
19:        lower_limit = −M
20:        upper_limit = M
21:        α = (upper_limit − lower_limit)/2
22:        counter = 0
23:    end while
24: end procedure
25: procedure ASK_ORACLE_IF_FEASIBLE(i, α)
26:    Ask the oracle if LP: minimize c^T x subject to x_i ≤ α, a_1^T x ≥ b_1, ... a_m^T x ≥ b_m is feasible
27:    if oracle says yes then
28:        return true
29:    else
30:        return false
31:    end if
32: end procedure
```

The outer loop has one iteration per component i of x. Each iteration of the inner while loop divides the range into half and asks the oracle if the new LP is feasible. If it is, then the middle of the range becomes the new upper end of the range, else it becomes the new lower end of the range. With each iteration, the range becomes half. So after $\log\left(\frac{M}{\epsilon}\right)$ iterations, the range becomes $\frac{2M}{2^{\log\left(\frac{M}{\epsilon}\right)}} = \frac{2M}{\frac{M}{\epsilon}} = 2\epsilon$. So after the inner iterations, the i^{th} component of the optimum solution of the LP will be within $\pm\epsilon$, for any given accuracy $\epsilon > 0$. After n iterations of the outer while loop, all n components of the LP solution will be within $\pm\epsilon$ of the correct value.

Question 4: Modified regression as a linear program ..... **[6]**

Regression is one of the common problems in data applications. We are given n vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ (in say d-dimensions), and real numbers $b_1, b_2, \ldots, b_n$. The goal is to find an $x \in \mathbb{R}^d$ such that $\langle \mathbf{a}_i, x \rangle \approx b_i$ for all i. In the usual least squares formulation, one tries to find an x so as to minimize $\sum_{i=1}^{n} |\langle \mathbf{a}_i, x \rangle - b_i|^2$.

Suppose we change the objective a little, and we wish to find an x so as to minimize

$$\sum_{i=1}^{n} |\langle \mathbf{a}_i, x \rangle - b_i|.$$

(i.e., the objective from earlier, but without the square). Prove that this optimization can be cast as a linear program of size polynomial in n, d. (You will receive partial credit if you come up with an exponential sized formulation.)

Let $R_i = |\langle \mathbf{a}_i, x \rangle - b_i|$ be the error for the i^{th} vector. We can construct the following linear program for the regression with the lowest error

$$
\begin{aligned}
&\text{minimize } R_1 + R_2 + \ldots + R_n \text{ subject to} \\
&\quad R_1 \geq 0 \\
&\quad R_2 \geq 0 \\
&\quad \ldots \\
&\quad R_n \geq 0 \\
&\quad R_1 \geq -\langle a_1, x \rangle + b_1 \\
&\quad R_1 \geq \langle a_1, x \rangle - b_1 \\
&\quad R_2 \geq -\langle a_2, x \rangle + b_2 \\
&\quad R_2 \geq \langle a_2, x \rangle - b_2 \\
&\quad \ldots \\
&\quad R_n \geq -\langle a_n, x \rangle + b_n \\
&\quad R_n \geq \langle a_n, x \rangle - b_n
\end{aligned}
$$

Question 5: Starbucks locations . . . . . **[12]**

Consider the following "Starbucks location" problem. We are given the locations of m strip malls, that are possible locations for opening a store. We are also given the addresses of n customers. Let d_{ij} be the distance between the i'th customer's address and the location of the jth strip mall. Let us assume that we are given as input this matrix d_{ij}.

Given a radius R, the goal is to find the smallest number of stores that need to be opened, so as to guarantee that every customer is at distance $\leq R$ to at least one of the opened stores.

Let us consider an optimization formulation that uses binary variables $\{y_j\}_{j=1}^{m}$, where y_j indicates if a store is to be opened at location j.

(a) **[3]** Write down the constraints one needs to place on the y_j's, and the objective function to optimize.

(b) **[5]** Suppose we "relax" the constraints $y_j \in \{0, 1\}$ to $0 \leq y_j \leq 1$, thus obtaining a linear program (that we can solve efficiently). Give an example (by drawing a picture of locations and addresses in 2D) where the obtained linear program has a strictly smaller optimum value than that of the original problem.

(c) **[4]** Suppose we solve the linear program and find that in the optimal solution, all the variables y_j are either 0 or are ≥ 0.5. Use this y to come up with a feasible way of opening stores such that (a) we satisfy the constraint that every user has an open store at distance $\leq R$, and (b) the number of stores opened is at most twice the optimum number.

(a) In order to model the requirement of maximum distance to a store from any customer address being $\leq R$, we can introduce a binary variable k_{ij} for the i^{th} customer and the j^{th} store. Consider the following linear program

$$\text{minimize } \sum_j y_j \text{ subject to} \tag{12}$$

$$\forall j \; y_j \in \{0,1\} \tag{13}$$

$$\sum_j y_j \geq 1 \tag{14}$$

$$\forall i,j \; k_{ij} \in \{0,1\} \tag{15}$$

$$\forall i \; \sum_j k_{ij} \geq 1 \text{ and } \forall i \; \sum_j k_{ij} \leq 1 \tag{16}$$

$$\forall i \; \sum_j d_{ij} y_j k_{ij} \leq R \tag{17}$$

(12) says that we are minimizing the total number of stores. This is subject to the following constraints

1. (13) says that the y_j corresponding to a store in strip mall location j should indicate that it be either not opened (corresponding to 0) or opened (corresponding to 1).
2. (14) says that at least 1 store should be opened.
3. (15) says that for customer i and strip mall location j, the binary variable k_{ij} will be 0 or 1.
4. (16) says that for any customer, at most one binary variable k_{ij} will be 1. This will be used to ensure that for customer i, only the nearest store will be considered in (17).
5. (17) enforces the constraint that for customer i, the sum of distances to all stores is within R. Due to variable k_{ij} described in (16), only one store will be considered per customer.

(b) Consider the following matrix that gives the distances d_{ij} from the i^{th} customer address to the location of the j^{th} strip mall:

$$\begin{bmatrix} 1 & 5 \\ 10 & 3 \end{bmatrix}$$

If one store is opened in each of the two strip malls, corresponding to $R = 3$, customer 1 will be located at a distance of 1 from strip mall location 1 and customer 2 will be located at a distance of 3 from strip mall location 2. This will be the optimum in the above case where $\sum_j y_j = 1 + 1 = 2$. If we "relax" the constraints $y_j \in \{0,1\}$ to $0 \leq y_j \leq 1$ and $k_{ij} \in \{0,1\}$ to $0 \leq k_{ij} \leq 1$, the following could be a solution to the linear program:

$$K = \begin{bmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \end{bmatrix} \text{ and } Y = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$$

Where K is the matrix for the variables k_{ij} and Y is the vector for the variables y_j. In this case $\sum_j y_j = 0.9 + 0.1 = 1$ and so has a smaller optimum.

(c) For the cases where $y_j = 0$, do not open a store at location j and open stores at locations corresponding to when $y_j \geq 0.5$.

a Since constraint (17) has been satisfied, it means that for all customers, there is at least one store at a distance of $\leq R$.

b The number of stores to be opened is given by $\sum_j y_j$. Suppose the count of $y_j \geq 0.5$ is n. In the worst case, all instances where $y_j \geq 0.5$, the value will be 0.5. In this case, the optimum number of stores to be opened is $0.5n$. Since a store is opened in all cases where $y_j \geq 0.5$, we will end up opening n stores, which is twice the optimum value of $0.5n$. This means that the number of stores opened is at most twice the optimum number corresponding to the worst case.