CSE 521: Design and Analysis of Algorithms I **Fall 2019**

Lecture 17: LP Applications – Linear Modeling, Planning, Optimization

Lecturer: Shayan Oveis Gharan *11/25/19*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

In this lecture we discuss several applications of linear programming. We start by discussing applications in linear modeling, then we will go to applications in planning and optimization.

17.1 Linear Modeling

In this part we discuss the idea of linear modeling. This idea has been used in a number of areas of computer science and finance. The rough idea is to disregard correlations between the parameters of a given problem and formulate a linear model for our problem at hand. For many real-world problems this gives a very good approximation of the optimum solution of the problem which involves the underlying correlations.

17.1.1 Example: Diet Problem

Suppose we have the problem of designing a diet that consists of certain quantities of various food groups, with quantities chosen to maximize some desired function of the food chosen while also satisfying constraints that are functions of properties of the food. The food groups are (v)eggies, (m)eat, (f)ruits, and (d)airy, with each food having the known characteristics of (p)rice, number of (c)alories, and amount of (h)appiness per unit of food (details in Table 17.1.1).

	veggies	meat	fruits	dairy
price	p_v	p_m	p_f	p_d
calories	c_v	c_m	c_f	c_d
happiness	h_v	h_m	h_f	h_d

Table 17.1: Constant-valued attributes for each food group.

In particular, we are assuming that one gram of veggies has price p_v, c_v calories and gives us h_v happiness. Let x_v, x_m, x_f, and x_d be the variables indicating the amount of grams from each of the four group that we should consume in a single day. Our objective is to choose x_v, x_m, x_f, x_d to maximize overall happiness, under the constraint that the total price must be less than 20 and the total number of calories must be less than 1500. We can formulate this as a linear program by writing

$$\begin{aligned} \text{maximize} \quad & x_v \cdot h_v + x_m \cdot h_m + x_f \cdot h_f + x_d \cdot h_d \\ \text{subject to} \quad & x_v \cdot p_v + x_m \cdot p_m + x_f \cdot p_f + x_d \cdot p_d \leq 20 \\ & x_v \cdot c_v + x_m \cdot c_m + x_f \cdot c_f + x_d \cdot c_d \leq 1500 \end{aligned} \tag{17.1}$$

Note that here we avoid correlations between the four categories in our model and we consider a linear model. This in particular means that our happiness from 100 grams of meat is independent of the dressing and other ingrediants of the food.

In addition, observe that the solution of the above program is not necessarily integral; for example in the optimal solution we may have to eat 2.5 grams of meat in a single day. We consider that being acceptable in our model.

17.1.2 Example: Support Vector Machine

The Support Vector Machine (SVM) is a classical machine learning model for linearly separating data points that belong to difference classes. Suppose we have m points $x_1, \ldots, x_m \in \mathbb{R}^d$ labeled "+" and n points $y_1, \ldots, y_n \in \mathbb{R}^d$ labeled "–". We want to find a linear separator (i.e. a hyperplane in $\mathbb{R}^d$) given by an orthogonal vector w and an offset b, such that all of the points $x_1, \ldots, x_m$ fall on one side of the separator (given by the constraint that $\langle w, x_i \rangle \geq b,\ i = 1, \ldots, m$) and all of the points $y_1, \ldots, y_n$ fall on the other side (given by $\langle w, y_j \rangle \leq b,\ j = 1, \ldots, n$). An example of such a set of points and a linear separator is given in Figure 17.1.2.

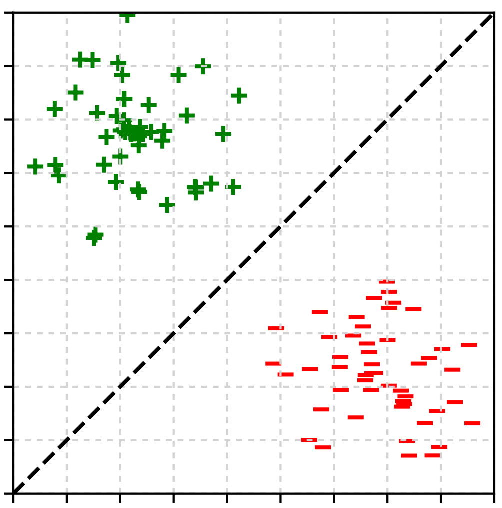

Figure 17.1: Example of data points and linear separator.

It turns out that this corresponds to a simple linear programming problem:

$$\langle w, x_i \rangle \geq b \forall 1 \leq i \leq m$$
$$\langle w, y_j \rangle \leq b \forall 1 \leq j \leq n$$

Note that the above linear program has no objective function. It is just a feasibility problem. There are many faster algorithms to solve the above problem that we are not going to discuss here, e.g., the Winnow's algorithm.

But the LP has many other advantages. Once we model a problem with a linear program we get to use more complex constraints or objective functions. For example, in many real-world instances of the support vector machine problem, a set of points may not be linearly separable, e.g., there could be some outliers. In this case, we can allow some small amount of error. We can model this problem as follows:

$$\begin{aligned} \underset{w,b,\epsilon,\delta}{\text{minimize}} \quad & \sum_{i=1}^{m} \epsilon_i + \sum_{j=1}^{n} \delta_j \\ \text{subject to} \quad & \langle w, x_i \rangle \geq b - \epsilon_i, \quad i = 1, \ldots, m \\ & \langle w, y_j \rangle \leq b + \delta_j, \quad j = 1, \ldots, n \\ & \epsilon_i, \delta_j \geq 0 \quad \forall i, j \end{aligned} \tag{17.2}$$

Now any in general we may want want to minimize some norm of the error parameters. In the above we minimize the L_1 norm. But it turns out that for a vector $x \in \mathbb{R}^n$ any norm is convex. So, we my want to use an L_2 norm or any infinity norm in the objective function depending on the application.

17.2 Planning/Decision-making

Consider the decision-making problem of choosing whether to pursue a Ph.D. after college or take a job in industry. Suppose we have a distribution X_1 of the potential salaries when obtaining a job after having done a Ph.D., and a distribution X_2 for salaries without having done a Ph.D. Suppose there is also a probability p of actually finishing a Ph.D. once it is started. We can reformulate the problem of deciding whether or not to get a Ph.D. as choosing the action that maximizes our expected gain: in this case, an industry job would lead to an expected gain of $\mathbb{E}X_1$, whereas pursuing a Ph.D. gives an expected gain of $p \cdot \mathbb{E}X_2 + (1-p) \cdot \mathbb{E}X_1$ (where the second term arises from the fact that we can still get an industry job even without completing the Ph.D.). If it is the case that $\mathbb{E}X_1 < p \cdot \mathbb{E}X_2 + (1-p) \cdot \mathbb{E}X_1$, then our expected gain of getting a Ph.D. is higher than getting an industry job, and going to grad school would be the optimal decision.

Markov Decision Processes Expanding on the previous section, let us define a model that consists of a set of states and actions, where each action takes us from one state to another and gains some reward. Specifically, if we are in state i and take action a, then we will go to state j with probability $P_a(i,j)$ and get a reward of $R_a(i,j)$. This setup defines a Markov Decision Process (MDP). Given a probability distribution for each pair of state/action, as well an associated reward function, we want to find an optimal policy that will maximize the cumulative reward.

There are two main directions to model the objective function:

Finite Horizon We run our MDP for a finite number of steps T, with the total reward defined as the sum of rewards across all time steps.

Infinite Horizon We run our MDP for an infinite number of steps, and the total reward is a discounted version of the sum of rewards across all time steps, such that rewards gained further into the future are discounted more.

17.2.1 Finite Horizon MDPs

Suppose we run our MDP for T steps and the rewards is the collective sum of the rewards for all steps. We can find the optimal policy by dynamic programming.

For an state i and time step $0 \leq t \leq T$, let $V_t(i)$, be the expected reward of an MDP that starts from state i at timestep t to time T. Assuming we know the value of $V_{t+1}(j)$ for all states j, we can define V_t recursively by choosing the action among all possible actions that maximizes our expected reward.

Firstly, observe that for any state i,

$$V_T(i) = 0.$$

This is because at time T we stop the chain so there is no more rewards to be collected. For $0 \leq t < T$ and state i we can write,

$$V_t(i) = \max_a \sum_j P_a(i,j)\,(R_a(i,j) + V_{t+1}(j)) = B(V_{t+1})(i) \tag{17.3}$$

Note that for an action a, the optimal expected reward if we take action a at state i at time t is

$$\sum_j P_a(i,j)(R_a(i,j) + V_{t+1}(j)).$$

Therefore, the optimal expected reward out of state i at time t is obtained by an action a the maximizes the above quantity (see (17.3)).

The function F that we defined in (17.3) is called the *Bellman operator*. Using this function, we can recursively compute V_t for all states and all time steps $T-1, T-2, \ldots, 1, 0$ in that order, relying on the fact that $V_T = 0$. In other words, we apply the Bellman operator, B, T times, to obtain V_0,

$$V_0 = \underbrace{B(B(\ldots B(}_{T \text{ times}}0))) \tag{17.4}$$

Where we used that $V_T = 0$.

To determine the optimal policy, we can simply replace the max on the right-hand side with an argmax. That is if we are at state i at time t we choose an action a which maximizes $\sum_j P_a(i,j)(R_a(i,j) + V_{t+1}(j))$.

The above algorithm takes (computation) time $O(Tn^2|A|)$ to calculate V_0 assuming that the chain has a total of n states and $|A|$ actions. Observe that this grows linearly with T, which is not ideal as computing all values for a very long sequence of steps would be computationally expensive.

17.2.2 Infinite Horizon

In this case, we assume the MDP runs for $T = \infty$ steps, and our total reward is now discounted by a rate $\gamma < 1$. Specifically, for every time step in the future that a reward is gained, it is multiplied by a factor of γ, so that a reward gained t steps into the future is scaled by γ^t. This can be thought of as adjusting for inflation, since rewards gained in the future are not worth as much as a reward gained at the current time step. Now, observe that since $\gamma < 1$ the collective rewards is always finite. A natural approach is to reduce this problem to the finite horizon case; intuitively by time $t \geq \frac{10 \log n}{1-\gamma}$ the discount is so huge that the collective reward is negligible from that time on. So, we just need to solve the finite horizon case for $T = \tilde{O})((1-\gamma)^{-1})$. Unfortunately for γ very close to 1, T is very large, and the algorithm is impractical. So, our idea is to use linear programming to find the optimal policy.

Firstly, suppose we apply discounting to the Finite Horizon case; then, we can write the corresponding Bellman operator as follows:

$$V_t(i) = \max_a \sum_j P_a(i,j)\,[R_a(i,j) + \gamma V_{t+1}(j)] = B_\gamma(V_{t+1})(i) \tag{17.5}$$

In other words, the optimal policy can be obtained by applying the B_γ operator an infinite number of times on the zero vector, i.e., the optimal reward is

$$V^* = \underbrace{B_\gamma(B_\gamma(\ldots}_{\infty \text{ times}}(0))\ldots)$$

Note that in the above equality it does not matter that we start from the 0 vector. If we start from any arbitrary vector V, we have

$$\underbrace{B_\gamma(B_\gamma(\ldots}_{\infty \text{ times}}(V))\ldots) = V^*$$

This is because the starting reward vector will be scaled down by $\gamma^\infty = 0$.

Here is the important observation about the optimal policy in the infinite Horizon case. Let $V_t^*(i)$ be the optimal expected reward when we start at state i at time t. We claim that for all i, t, $V_t^*(i) = V_{t+1}^*(i)$. This is because no matter when we start the chain, we will always run for an infinite number of steps. It follows from this observation that the optimal reward, V^*, is constant across all times; furthermore, for each state i (and any time t) the optimum action is an action a where

$$V^*(i) = \sum_j P_a(i,j)(R_a(i,j) + \gamma V^*(j)).$$

Having this in mind, we claim that the following linear program gives the optimum expected reward at any state i.

$$V \geq B_\gamma(V) \tag{17.6}$$

In the above program, the constraint $V \geq F_\gamma(V)$ is indeed a vector constraint; so in particular, for any state i we have a constraint

$$V(i) \geq \max_a \sum_j P_a(i,j)\left[R_a(i,j) + \gamma V(j)\right].$$

Note that this is a feasibility program, as there is no objective function. In our main theorem we show that any feasible solution of the program is the optimum policy. Also, observe that maximum of linear functions is a convex function and we can upper bound a convex function. In this particular case we can implement the above convex program by a linear program by simply having the following inequality for every possible action a,

$$V(i) \geq \sum_j P_a(i,j)[R_a(i,j) + \gamma V(j)].$$

We prove the following theorem

Theorem 17.1. *The optimal solution of* (17.6) *gives the expected rewards of the optimal policy.*

First recall that since $V^* = B_\gamma(V^*)$, V^* is a feasible solution of the linear program. So, indeed we wanted to write the constraint $V = B_\gamma(V)$ in the LP. But this is not a convex constraint.

We use the following fact to prove the theorem.

Fact 17.2. *If $U \geq V$, then $B_\gamma(U) \geq B_\gamma(V)$*

Proof. Based on its definition, B_γ is a monotone operator. This means that if the value of V is increased for every state i, $B_\gamma(V)$ can only increase in value. Since $U \geq V$ element-wise for every state, this implies that $B_\gamma(U) \geq B_\gamma(V)$. □

Using these two facts, we can now prove Theorem 17.1:

Proof of Theorem 17.1. Let V be a feasible solution to the linear program given in (17.6), so that $V \geq B_\gamma(V)$. By Fact 17.2, we have

$$B_\gamma(V) \geq B_\gamma(B_\gamma(V))$$

By another application of Fact 17.2 to the above inequality we have

$$B_\gamma(B_\gamma(V)) \geq B_\gamma(B_\gamma(B_\gamma(V))).$$

Following this line of reasoning we get,

$$V \geq B_\gamma(V) \geq B_\gamma(B_\gamma(V)) \geq \cdots \geq B_\gamma^\infty(V) = V^*$$

This means that $V \geq V^*$. But since V^* is the optimum reward, all of the above inequalities indeed must be equalities. $\square$

17.3 Optimization

Consider an optimization problem where we are trying to find a solution with minimum cost among a set of feasible solutions. We say an algorithm, ALG, gives an α-approximation for the problem if for any possible input to the problem, we have

$$\frac{\text{cost(ALG)}}{\text{cost(OPT)}} \leq \alpha \tag{17.7}$$

Here, OPT denotes the optimum solution to the problem.

To prove that a given algorithm is an α-approximation, it is sufficient to find a lower-bound for cost(OPT), and then prove that the ratio between cost(ALG) and this lower-bound for any input is upper-bounded by α.

17.3.1 Example: Vertex Cover

Here, we give an application of linear programming in designing an approximation algorithm for a graph problem called vertex cover. We will design a 2-approximation algorithm. This is the best known result for the vertex cover problem. It is a fundamental open problem to beat the factor 2 approximation for the vertex cover problem. In the next lecture we will discuss a generalization of vertex cover called the set cover problem and we see some applications.

Given a graph $G = (V, E)$, we want to find a set $S \subset V$ such that every edge in E is incident to at least one vertex in S. Obviously, we can let $S = V$. But, here among all such sets S we want to choose a one of minimal cost, where cost(S) is defined as $\sum_{i \in S} c_i$ if every vertex i has associated cost c_i, and $|S|$ if vertices do not have any cost.

In the first step we write a (integer) program which characterizes the optimum solution. Then, we use this program to give a lower bound on the optimum solution. We define this problem with a set of variables $x_i \; \forall\, i \in V$, where x_i is defined as

$$x_i = \begin{cases} 1 & i \in S \\ 0 & i \notin S \end{cases} \tag{17.8}$$

Our constraint that every edge must be incident to at least one vertex in S can be written as $x_i + x_j \geq 1 \; \forall\, i \sim j \in E$. So, the question is to find values for all x_i's that minimize the cost of the set S subject to the aforementioned constraint. This can be defined as the following optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i \in V} c_i x_i \\ \text{subject to} \quad & x_i + x_j \geq 1, \; \forall\, i \sim j \in E \\ & x_i \in \{0, 1\}, \; \forall\, i \in V \end{aligned} \tag{17.9}$$

Observe that the optimum solution of the above program is exactly equal to the optimum set cover. Note that this is not a linear program, since we have that $x_i \in \{0,1\}$ for every vertex i, rather than allowing x_i to be a continuous-valued variable. Since the vertex cover problem is NP-hard in general, we do not expect to ever find a general solver to efficiently solve the above integer program. However, there are commercial integer programming solver that work great in practice. They solve a set of linear inequality subject to the each of the underlying variables being 0/1. For many practical applications these program actually find the optimum solution very fast. So, one should always keep them in mind if we are trying to solve an optimization problem in practice.

We can relax the above (integer) program by replacing the integer constraint with the constraint that $0 \leq x_i \leq 1 \; \forall i \in V$. This turns the problem into a linear program. Since this is optimizing over a set of x_i's that includes the optimum set cover, the optimal value of this linear program will be less than or equal to the optimal value of the set cover problem, i.e. OPT LP $\leq$ OPT. The resulting linear program can be written as

$$\begin{aligned} \min \quad & \sum_{i \in V} c_i x_i \\ \text{subject to} \quad & x_i + x_j \geq 1, \; \forall i \sim j \in E \\ & 0 \leq x_i \leq 1, \; \forall i \in V \end{aligned} \tag{17.10}$$

Suppose we have an optimal solution of the above program. We want to round this solution into a set cover such that the cost of the cover that we produce is within a small factor of the cost of the LP solution.

The idea is to ue a simple thresholding idea: For each vertex i, if $x_i \geq 0.5$, then we add i to S, otherwise we don't include i in S.

Claim 17.3. *For any solution x of linear program (17.10), the resulting set S, is a vertex cover*

Proof. For a feasible solution x to the linear program, we know that $x_i + x_j \geq 1 \; \forall i \sim j \in E$. This means that for every edge $i \sim j$, at least one of x_i, x_j is at least 0.5.Therefore, for any edge $i \sim j$ at least one of i, j is in S. So, S is a vertex cover. □

Claim 17.4. *For any solution x of linear program (17.10) the resulting set S satisfies*

$$\sum_{i \in S} c_i \leq 2 \sum_i c_i x_i = OPT\ LP.$$

This implies that the above algorithm is a 2 approximation for the vertex cover problem.

Proof.

$$\sum_{i \in S} c_i = \sum_{i: x_i \geq 0.5} c_i \leq \sum_{i: x_i \geq 0.5} 2 c_i x_i \leq \sum_i c_i x_i.$$

□

Note that in the worst case $x_i = 0.5$ for all vertices i and the above claim is tight.