CS221 Fall 2018 Homework 1

SUNet ID: 05794739
Name: Luis Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) The problem asks us to find the value of θ that minimizes the function:

$$f(\theta) = \frac{1}{2}\sum_{i=1}^{n} w_i(\theta - x_i)^2$$

We can compute the above by simply taking the derivate of the function with respect to θ and solving for θ when equal to 0.

$$\begin{aligned}\frac{df}{d\theta} &= \sum_{i=1}^{n} w_i(\theta - x_i) && \text{(using product rule } (uv)' = u'v + uv')\\ &= 0 \implies \theta = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}\end{aligned}$$

We note that $\frac{d^2 f}{d\theta^2} = \sum_{i=1}^{n} w_i$, so the above solution is a minimum if and only if $\sum_{i=1} w_i > 0$. This is guaranteed if $w_i > 0, \forall i$. However, if we have some $w_i < 0$, then we could run into an issue where the function does not have a minimum.

(b) We can show what we want rather directly. In particular, we will show that $f(\boldsymbol{x}) \geq g(\boldsymbol{x})$.

$$\begin{aligned}f(\boldsymbol{x}) &= \sum_{i=1}^{d} \max_{s\in\{1,-1\}} sx_i && \text{(definition of } f)\\ &= \max_{\boldsymbol{s}\in\{1,-1\}^d} \boldsymbol{s}^T\boldsymbol{x} && \text{(definition of dot product)}\\ &\geq \max_{\boldsymbol{s}\in\{[1]^d,[-1]^d\}} \boldsymbol{s}^T\boldsymbol{x} && (\boldsymbol{s} \text{ can take on fewer values now})\\ &= \max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i && \text{(definition of dot product)}\\ &= g(\boldsymbol{x})\end{aligned}$$

(c) Let X be a random variable representing the expected number of points you'll receive from playing this game and D be a random variable representing the result of the die

roll. Then we can solve this analytically. We first condition on the value of the fist dice roll.

$$\begin{aligned} E[X] = &E[X \mid D = 1]P(D = 1) \\ &+ E[X \mid D = 2]P(D = 2) \\ &+ E[X \mid D \in \{3, 4, 5\}]P(D \in \{3, 4, 5\}) \\ &+ E[X \mid D = 6]P(D = 6) \end{aligned}$$

Then we note the following identities:

$$E[X \mid D = 1] = 0 \qquad \text{(we will receive no points)}$$

$$E[X \mid D \in \{3, 4, 5\}] = E[X] \qquad \text{(the game is memoryless, so we can ignore the roll)}$$

$$E[X \mid D = 2] = (E[X] - a)$$

(the game is memoryless, and whatever points we expect to get, we'll lose a of them)

$$E[X \mid D = 6] = (E[X] + b)$$

(similar reasoning to above, but we will also gain b points)

With the above together, we have:

$$\begin{aligned} E[X] &= \frac{E[X] - a}{6} + \frac{E[X]}{2} + \frac{E[X] + b}{6} \\ &= \frac{5}{6}E[X] + \frac{b - a}{6} \\ \implies E[X] &= b - a \end{aligned}$$

(d) As the hint indicates, we can calculate the value of p that maximizes $L(p)$ by taking the derivate of $\log L(p)$ and setting it to zero (further nothing that $L(p)$ is concave, and therefore the critical point is a maximum).

$$\begin{aligned} \frac{d}{dp} \log L(p) &= \frac{d}{dp} \left[4 \log p + 3 \log(1 - p)\right] \\ &= \frac{4}{p} - \frac{3}{1 - p} \\ &= 0 \\ \implies 4 - 4p &= 3p \\ \implies p &= \frac{4}{7} \end{aligned}$$

The intuitive interpretation of this value of p is that this is the probability of the coin which we used of landing heads. Given the data, it appears to be slightly biased in favor of heads.

(e) We're given the function $f(\boldsymbol{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})^2 + \lambda||w||_2^2$ and we wish to compute $\nabla f(\boldsymbol{w})$. We can do it by parts, first.

$$\begin{aligned}\frac{\partial}{\partial w_k}||\boldsymbol{w}||_2^2 &= \frac{\partial}{\partial w_k}\sum_{k=1}^{d} w_k^2 \\ &= 2w_k \qquad \text{(all terms except } w_k \text{ are zero)}\end{aligned}$$

We therefore have $\nabla\lambda||\boldsymbol{w}||_2^2 = 2\lambda\boldsymbol{w}^T$. Continuing:

$$\begin{aligned}\frac{\partial}{\partial w_k}\left[\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})^2\right] &= \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial}{\partial w_k}\left[\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w}\right]^2 \\ &= 2\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})\frac{\partial}{\partial w_k}[\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w}] \\ &\qquad \text{(chain rule)} \\ &= 2\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})[\boldsymbol{a}_i - \boldsymbol{b}_j]_k^T\end{aligned}$$

We therefore have the result that:

$$\begin{aligned}\nabla f(\boldsymbol{w}) &= 2\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})[\boldsymbol{a}_i - \boldsymbol{b}_j]^T + 2\lambda\boldsymbol{w}^T \\ &= 2\boldsymbol{w}^T\left[\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i - \boldsymbol{b}_j)(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\right] + 2\lambda\boldsymbol{w}^T\end{aligned}$$

Problem 2

(a) We first note that there are a total of $\binom{n^2}{2} = O(n^4)$ locations to place a rectangle in an $n \times n$ grid (think about it as if you're choosing the top-left and bottom-right corners, which will fully determine an axis-aligned rectangle). To fully determine a face, we need to place 2 (eyes) + 2 (ears) + 1 (nose) + 1 (mouth) = 6 rectangles total. To simplify things (this won't affect the final answer), we assume that the same rectangle can use used for multiple face parts. We therefore have a total of $O(n^4) \times O(n^4) \times O(n^4) \times O(n^4) \times O(n^4) \times O(n^4) = O(n^{24})$ choices of 6 rectangles each choise defining one face for a total of $O(n^{24})$ possible faces.[1]

(b) This is essentially a DP problem. First, define $T(i, j)$ to be the minimum cost for reaching the lower-right corner (n, n) given that we're currently at position (i, j) for

[1]Technically, we're double counting the eyes and the ears here – however, note that this will add at most a constant factor of 4, so it does not affect the big-O solution.

$1 \leq i, j \leq n$. We can define this recursively as follows:

$$\begin{aligned} T(n,n) &= c(n,n) \\ T(i,n) &= c(i,n) + T(i+1,n) \\ T(n,j) &= c(n,j) + T(n,j+1) \\ T(i,j) &= c(i,j) + \min\{T(i,j+1), T(i+1,j)\} \end{aligned}$$

To answer the question asked in the problem statement, we would simply need to return $T(1,1)$. We can memoize intermediate results and thereby achieve a running time of $O(n^2)$ – n^2 possible inputs, each taking contant time to compute – with space complexity of $O(n^2)$. We can further minimize the space-complexity to be O(n) if we do an iterative matrix-filling approach in an intelligent way, but this is not required for this problem.

(c) We can compute this rather directly by thinking of the problem slightly differently. Basically, there are n steps. Each "way" to reach the top can be encoded uniquely by a binary string of length $n-1$ indicating whether or not we step on the i-th step or not on our way to the n-th step. For the example give, when $n = 4$ we have $000, 001, 010, 100, 011, 101, 110, 111$ which respectively encode (1) step on 4, (2) take two steps and step on (a) 3,4, (b) 2,4, (c) 1,4, (3) take three steps and step on (a) 2,3,4, (b) 1,3,4, (c) 1,2,4, and (4) take four steps by stepping on all the steps.

As such, the total number of ways is simply $2^{\max\{0,n-1\}}$, where we take $\max\{0, n-1\}$ to handle the $n = 0$ case nicely.

(d) We can device such an algorithm for computing $f(\boldsymbol{w})$ by following the hint. First, we note that $\lambda||\boldsymbol{w}||_2^2$ can be computed in $O(d)$ time (squaring and summing d values). So we just need to find an efficient way to compute the first term. We note:

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_i^T\boldsymbol{w})^2 &= \sum_{i=1}^{n}\sum_{j=1}^{n}[(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\boldsymbol{w}]^2 \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}[(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\boldsymbol{w}]^T(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\boldsymbol{w} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{w}^T(\boldsymbol{a}_i - \boldsymbol{b}_j)(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\boldsymbol{w} \\ &= \boldsymbol{w}^T\left[\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i - \boldsymbol{b}_j)(\boldsymbol{a}_i - \boldsymbol{b}_j)^T\right]\boldsymbol{w} \end{aligned}$$

where the inner double summation will result in a $d \times d$ matrix that does not depend on w. Note that we focus on finding an efficient method of computing this value. First,

let use define $\boldsymbol{A}$ and $\boldsymbol{B}$ as the $d \times n$ matrices consisting of the $\{\boldsymbol{a}_i\}$ and $\{\boldsymbol{b}_i\}$ as column vectors. Then we note the following:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i - \boldsymbol{b}_j)(\boldsymbol{a}_i - \boldsymbol{b}_j)^T = n\sum_{i=1}^{n}\boldsymbol{a}_i\boldsymbol{a}_i^T + n\sum_{j=1}^{n}\boldsymbol{b}_j\boldsymbol{b}_j^T - \sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{a}_i\boldsymbol{b}_j^T - \sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{b}_j\boldsymbol{a}_i^T$$

Note that the first two terms can be computed directly as defined in $O(nd)$ time. At first glance, the last two terms appear to require $O(n^2d)$ time (as defined). However, we note that:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{a}_i\boldsymbol{b}_j^T = \boldsymbol{A}^T\boldsymbol{B}$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{b}_j\boldsymbol{a}_i^T = \boldsymbol{B}^T\boldsymbol{A}$$

As such, we can immediately note that we can actually compute the above values in $O(nd^2)$ time using matrix multiplication.

We have now pre-computed:

$$\boldsymbol{X} = \sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i - \boldsymbol{b}_j)(\boldsymbol{a}_i - \boldsymbol{b}_j)^T$$

where $\boldsymbol{X}$ is a $d \times d$ matrix in $O(nd^2)$ time. Thefore, now for any vector $\boldsymbol{w}$, we can just need to compute:

$$f(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{X}\boldsymbol{w} + \lambda||\boldsymbol{w}||_2^2$$

The second term is $O(d)$ and the first term can be computed in $O(d^2)$ time. We therefore have an $O(d^2)$ time algorithm for computing f.