CS221 Fall 2015 Homework 1

SUNet ID: jaigupta
Name: Jai Gupta

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)

$$\begin{aligned} f'(\theta) &= \frac{1}{2}\sum_{i=1}^{n} w_i(\theta - x_i) * 2 \\ &= \sum_{i=1}^{n} w_i(\theta - x_i) \end{aligned} \tag{1}$$

Putting $f'(\theta) = 0$

$$\sum_{i=1}^{n} w_i(\theta - x_i) = 0$$

$$\begin{aligned} \sum_{i=1}^{n} w_i\theta &= \sum_{i=1}^{n} w_i x_i \\ \theta * \sum_{i=1}^{n} w_i &= \sum_{i=1}^{n} w_i x_i \\ \theta &= \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \end{aligned} \tag{2}$$

Additionally, $f''(\theta) = \sum_{i=1}^{n} w_i$, which is always positive since the all w_i values are positive. Hence, the above found value of θ represents a minima.

The above value of θ can be thought of as the weighted mean of the x_i values.

If w_i values can be negative, then the existence of mimima is conditioned on the above mentioned value of $f''(\theta) = \sum_{i=1}^{n} w_i$. If the value is:

- positive $\rightarrow$ minima exists at the θ found in equation 2.
- zero $\rightarrow$ the equation represents a hyper place, neither minima nor maxima exist.
- negative $\rightarrow$ maxima exists at the θ found in 2. Minima does not exist.

(b) For $f(\mathbf{x}) = \sum_{i=1}^{d} max_{s\epsilon\{1,-1\}} s x_i$, we can set s to 1 whenever x_i is positive and -1 for negative values of x_i. Either of them will work if x_i is zero. Hence $f(\mathbf{x})$ will be $\sum_{i=1}^{d} |x_i|$.

For $g(\mathbf{x}) = max_{s\epsilon\{1,-1\}} \sum_{i=1}^{d} sx_i$, we can rewrite it as $g(\mathbf{x}) = max_{s\epsilon\{1,-1\}} s \sum_{i=1}^{d} x_i$. Since we can only have two values of s, the above equation can be simplified further simplified to $g(\mathbf{x}) = |\sum_{i=1}^{d} x_i|$.

If all the x_i values have same sign, clearly $f(\mathbf{x})$ will have the same value as $g(\mathbf{x})$. But if they don't, $f(\mathbf{x})$ adds up absolute values of entries and hence they don't cancel each other. But in $g(\mathbf{x})$, they will cancel each other since the absolute value operation is delayed to be computed after taking the sum thereby reducing the values of $g(\mathbf{x})$ compared to $f(\mathbf{x})$. Therefore, $f(\mathbf{x}) \geq g(\mathbf{x})$ for all values of $\mathbf{x}$

(c) Let E denote the expected value. Then

$$\begin{aligned} E &= \frac{1}{6} * 0 + \frac{1}{6}(-a + E) + \frac{1}{6}(b + E) + \frac{3}{6}E \\ E &= \frac{b-a}{6} + \frac{5E}{6} \\ \frac{E}{6} &= \frac{b-a}{6} \\ E &= b - a \end{aligned}$$

(d) Let $E(k,p)$ denote the expectation when rolling for at max k times and then taking 15 points. Here p denotes the probability of success (rolling an odd number). Then,

$$E(k,p) = 3 + p * 3 + p^2 * 3 .... p^{k-1} * 3 + p^k * 15 \quad = 3\frac{1-p^k}{1-p} + 15p^k$$

Even though k is not a continuous variable, we can still try differentiating it to see the slope of $E(k,p)$ with respect to k.

$$E'(k,p) = -\frac{3p^k ln(p)}{1-p} + 15p^k ln(p) \quad = p^k ln(p)(15 - \frac{3}{1-p})$$

In the first part, probability of success (probability of rolling an odd number) is $\frac{3}{5}$. $E'(k,p)$ is $\frac{15}{2} ln(\frac{3}{5})(\frac{3}{5})^k$ which is always negative since $ln(\frac{3}{5})$ is negative. Hence, the expected value reduces with k and the best strategy will be to have $k = 0$, i.e., not play any turn and take 15 points.

If $p = 1$, then $E(k,p)$ is a strictly increasing function of k ($E(k,p) = 3k + 15$) and hence we should just keep playing infinitely many number of times.

(e)

$$\begin{aligned} L(p) &= p^4(1-p)^3 \\ log(L(p)) &= 4log(p) + 3log(1-p) \\ \frac{\partial log(L(p))}{\partial p} &= \frac{4}{p} - \frac{3}{1-p} \end{aligned}$$

Putting $\frac{\partial log(L(p))}{\partial p} = 0$

$$\begin{aligned}
\frac{4}{p} - \frac{3}{1-p} &= 0 \\
4(1-p) &= 3p \\
4 &= 7p \\
p &= \frac{4}{7}
\end{aligned}$$

Additionally,

$$\frac{\partial^2 log(L(p))}{\partial p^2} = -\frac{4}{p^2} - \frac{3}{(1-p)^2}$$

Hence, second order derivative is negative for $p = \frac{4}{7}$, which means $\frac{4}{7}$ is a maxima.

Intuitively, one can think of the probability of $\frac{4}{7}$ to represent the fact that we expect head to occur 4 times in a sequence of 7 trials. If we increase this probability, we would expect more heads, and vice-versa.

(f)

$$\begin{aligned}
f(\mathbf{w}) &= \sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})^2 + \lambda||\mathbf{w}||_2^2 \\
\frac{\partial f(\mathbf{w})}{\partial w_k} &= \sum_{i=1}^{n}\sum_{j=1}^{n}2(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})\frac{\partial(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})}{\partial w_k} + \frac{\partial(w_1^2 + w_2^2 ... + w_k^2 + ...)}{\partial w_k} \\
\frac{\partial f(\mathbf{w})}{\partial w_k} &= \sum_{i=1}^{n}\sum_{j=1}^{n}2(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})(a_{ik} - b_{jk}) + 2w_k \\
\nabla_{\mathbf{w}} f(\mathbf{w}) &= \sum_{i=1}^{n}\sum_{j=1}^{n}2(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})(\mathbf{a}_i - \mathbf{b}_j) + 2\mathbf{w} \\
\nabla_{\mathbf{w}} f(\mathbf{w}) &= 2\mathbf{w} + 2\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{a}_i^T - \mathbf{b}_j^T)\mathbf{w}(\mathbf{a}_i - \mathbf{b}_j)
\end{aligned}$$

Problem 2

(a) A rectangle is composed of 2 horizontal and 2 vertical lines. Since we are only considering axis aligned rectangles, there are only n choices for horizontal lines, and n choices for vertical lines. The number of ways to choose a pair of lines is ${}^nC_2 + n$ where the first term is for choosing two distinct lines, and the second one for the case when the two lines are the same. Essentially there are $O(n^2)$ ways for each of horizontal and vertical lines giving a total of $O(n^4)$ choices for selecting a rectangle.
Since we need to select 6 such rectangles we will have $6 * O(n^4) = O(n^4)$ ways of selecting the combination of component rectangles to check for a face.

(b)
- Let $J(i,j)$ be the minimum cost of reaching $(i,\ j)$.
- We can reach $(i,\ j)$ from $(i-1,\ j)$ or $(i-1,\ j)$ only (single step down or right).

$$J(i,j) = min(J(i-1,j), J(i,j-1)) + c(i,j)$$

- This creates a recursive DAG structure to solve for using dynamic programming due to clear overlapping subproblems.
- Base cases in the above solution recursive solution are:

$$J(1,1) = c(1,1)$$
$$J(i,1) = J(i-1,1) + c(i,1);\ \text{If } i \neq 1$$
$$J(1,j) = J(1,j-1) + c(1,j);\ \text{If } j \neq 1$$

- Since dynamic programming reuses previously calculated values, constant time is needed to calculate $J(i,j)$ from previously evaluated sub problems. There are n^2 cells in the grid, thereby leading to a total time complexity of $O(n^2)$.

(c) This problem can also be converted to a dynamic programming approach by considering subproblems.

- Given a list of steps taken to cover n stairs as $s_1, s_2, ...s_k$ where s_i represents the size of the i_{th} step and they sum to n, we can remove the last step to find a way to cover $n - s_k$ steps.
- Thus removing the last step of every possible way gives us a unique way of reaching a previous stair (sub problem). Hence, if W(n) represents the number of ways of reaching stair n, then

$$W(n) = \sum_{i=0}^{n-1} W(i)$$

- Base case: $W(0) = 1$
- Once we have this recursive equation, we can additionally prove that $W(n) = 2^{n-1} \forall n \geq 1$
- We provide proof by induction:
  - Base case: $W(1) = \sum_{i=0}^{0} W(i) = W(0) = 1 = 2^{1-1}$, hence base case works.
  - Let it be true for all integers less than n, then:

$$\begin{aligned} W(n) &= \sum_{i=0}^{n-1} W(i) \\ &= W(0) + W(1) + W(2)....W(n-1) \\ &= 1 + 2^0 + 2^1 + ...2^{n-2} \\ &= 1 + (2^{n-1} - 1) \\ &= 2^{n-1} \end{aligned}$$

– Hence, assuming it to be true for integers less than n, we were able to prove it to work for n as well. Therefore, it works for all n ≥ 1.

(d)

$$f(\mathbf{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{a}_i^T\mathbf{w} - \mathbf{b}_j^T\mathbf{w})^2 + \lambda||\mathbf{w}||_2^2$$
$$f(\mathbf{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i^T - \mathbf{b}_j^T)\mathbf{w})^2 + \lambda||\mathbf{w}||_2^2$$

Note that the term being squared is a scalar, and hence can be the complete expression can be rewritten as:

$$f(\mathbf{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{w}^T(\mathbf{a}_i - \mathbf{b}_j)(\mathbf{a}_i^T - \mathbf{b}_j^T)\mathbf{w} + \lambda||\mathbf{w}||_2^2$$
$$f(\mathbf{w}) = \mathbf{w}^T(\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{a}_i - \mathbf{b}_j)(\mathbf{a}_i^T - \mathbf{b}_j^T))\mathbf{w} + \lambda||\mathbf{w}||_2^2$$

The following expression can be pre-calculated:

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

Time complexity of the above expression would be:

$$O(n)O(d^2) + (2 * O(nd^2) + O(d^2)) + O(nd^2) + (2 * O(nd^2) + O(d^2)) = O(nd^2)$$

Here $\mathbf{r}$ is a matrix of size d^2. Given a input matrix $\mathbf{w}$, we need to calculate:

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

This expression can be calculated in $O(d^2)$