cs221 Assignment #1: foundations

Problem 1. Optimization and probability

a. Let $x_1, \ldots, x_n$ be real numbers representing positions on a number line. Let $w_1, \ldots, w_n$ be positive real numbers representing the importance of each of these positions. Consider the quadratic function: $f(\theta) = \frac{1}{2}\sum_{i=1}^{n} w_i(\theta - x_i)^2$. What value of θ minimizes $f(\theta)$? What problematic issues could arise if some of the w_i's are negative?

Note: You can think about this problem as trying to find the point θ that's not too far away from the x_i's. Over time, hopefully you'll appreciate how nice quadratic functions are to minimize.

sol)

$$\begin{aligned}\frac{\partial}{\partial\theta} f(\theta) &= \frac{\partial}{\partial\theta}\frac{1}{2}\sum_{i=1}^{n} w_i(\theta - x_i)^2 \\ &= \sum_{i=1}^{n} w_i(\theta - x_i) = \theta\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i x_i = 0 \\ \therefore \hat{\theta} &= \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}\end{aligned}$$

As some of the w_i's are negative, the denominator could go to zero and $\hat{\theta}$ can go to infinity.

b. In this class, there will be a lot of sums and maxes. Let's see what happens if we switch the order. Let $f(\mathbf{x}) = \sum_{i=1}^{d} \max_{s\in\{1,-1\}} sx_i$ and $g(\mathbf{x}) = \max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i$, where $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is a real vector. Does $f(\mathbf{x}) \leq g(\mathbf{x})$, $f(\mathbf{x}) = g(\mathbf{x})$, or $f(\mathbf{x}) \geq g(\mathbf{x})$ hold for all $\mathbf{x}$? Prove it.

Hint: You may find it helpful to refactor the expressions so that they are maximizing the same quantity over different sized sets.

sol)

$$f(\mathbf{x}) = \sum_{i=1}^{d} \max_{s\in\{1,-1\}} sx_i = \sum_{i=1}^{d} |x_i| \text{ and } g(\mathbf{x}) = \max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i = \left|\sum_{i=1}^{d} x_i\right|,$$

Therefore by the triangle inequality, $f(\mathbf{x}) \geq g(\mathbf{x}$ for all $\mathbf{x}$.

c. Suppose you repeatedly roll a fair six-sided die until you roll a 1 (and then you stop). Every time you roll a 2, you lose a points, and every time you roll a 6, you win b points. You do not win or lose any points if you roll a 3, 4, or a 5. What is the expected number of points (as a function of a and b) you will have when you stop?

Hint: it is recommended to think of defining a recurrence.

sol)

$$\begin{aligned}\mathbb{E}(\#\text{ of trials to end casting a die}) &= 1\times\frac{1}{6} + 2\times\frac{5}{6}\times\frac{1}{6} + 3\times\frac{5}{6}\times\frac{5}{6}\times\frac{1}{6} + \ldots \\ &= \sum_{n=1}^{\infty} n\cdot\left(\frac{5}{6}\right)^{n-1}\cdot\frac{1}{6} = 6\end{aligned}$$

Hence, on average we have 5 drawing chances (before the last draw) to get or lose the points and while getting or losing the points, we only care 5 sides. (since we only care the process until appearing a 1, we can neglect the possibility of appearing 1 WLOG)

Therefore, in each draw, expected number of point is $b\times\frac{1}{5} - a\times\frac{1}{5} = \frac{b-a}{5}$, total expected points $= 5\times\frac{b-a}{5} = b - a$.

d. Suppose the probability of a coin turning up heads is $0 < p < 1$, and that we flip it 7 times and get $\{\mathrm{H}, \mathrm{H}, \mathrm{T}, \mathrm{H}, \mathrm{T}, \mathrm{T}, \mathrm{H}\}$. We know the probability (likelihood) of obtaining this sequence is $L(p) = pp(1-p)p(1-p)(1-p)p = p^4(1-p)^3$. What value of p maximizes $L(p)$? What is an intuitive interpretation of this value of p?

Hint: Consider taking the derivative of $\log L(p)$. You can also directly take the derivative of $L(p)$, but it is cleaner and more natural to differentiate $\log L(p)$. You can verify for yourself that the value of p which maximizes $\log L(p)$ must also maximize $\log L(p)$ (you are not required to prove this in your solution).

sol)

$$\begin{aligned}\frac{\partial}{\partial p}\log L(p) &= \frac{\partial}{\partial p}\log(p^4(1-p)^3) = \frac{\partial}{\partial p}4\log p + 3\log(1-p) \\ &= \frac{4}{p} + \frac{3}{1-p} = \frac{4\cdot(1-p)+3\cdot p}{p(1-p)} = \frac{4-7p}{p(1-p)} = 0\end{aligned}$$

$\therefore \underset{p}{\operatorname{argmax}} \log L(p) = \frac{4}{7} = \underset{p}{\operatorname{argmax}} L(p)$

Also, it is natural to think p as $4/7$ since heads appear 4 times among 7 coin tossing trials.

e. Let's practice taking gradients, which is a key operation for being able to optimize continuous functions. For $\mathbf{w} \in \mathbb{R}^d$ (represented as a column vector) and constants $\mathbf{a}_i, \mathbf{b}_j \in \mathbb{R}^d$ (also represented as column vectors) and $\lambda \in \mathbb{R}$, define the scalar-valued function

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

where the vector is $\mathbf{w} = (w_1, \ldots, w_d)^\mathsf{T}$ and $\|\mathbf{w}\|_2 = \sqrt{\sum_{k=1}^{d} w_k^2}$ is known as the L_2 norm. Compute the gradient $\nabla f(\mathbf{w})$.

Recall: the gradient is a d-dimensional vector of the partial derivatives with respect to each w_i:

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\mathsf{T}$$

If you're not comfortable with vector calculus, first warm up by working out this problem using scalars in place of vectors and derivatives in place of gradients. Not everything for scalars goes through for vectors, but the two should at least be consistent with each other (when $d = 1$). Do not write out summation over dimensions, because that gets tedious.

sol)

$$\begin{aligned}f(\mathbf{w}) &= \sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{a}_i^\mathsf{T}\mathbf{w} - \mathbf{b}_j^\mathsf{T}\mathbf{w})^2 + \lambda\|\mathbf{w}\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w})^2 + \lambda\sum_{m=1}^{d} w_m^2 \\ \frac{\partial f(\mathbf{w})}{\partial w_k} &= \sum_{i=1}^{n}\sum_{j=1}^{n} 2(\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w} \times \frac{\partial(\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w}}{\partial w_k} + 2\lambda w_k \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} 2(\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w}(\mathbf{a}_{ik} - \mathbf{b}_{jk}) + 2\lambda w_k \\ \therefore \nabla f(\mathbf{w}) &= \nabla\left(\sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w})^2 + \lambda\|\mathbf{w}\|_2^2\right) = 2\sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w}(\mathbf{a}_i - \mathbf{b}_j)) + 2\lambda\mathbf{w}\end{aligned}$$

Problem 2: Complexity

a. Suppose we have an image of a human face consisting of $n \times n$ pixels. In our simplified setting, a face consists of two eyes, two ears, one nose, and one mouth, each represented as an arbitrary axis-aligned rectangle (i.e. the axes of the rectangle are aligned with the axes of the image). As we'd like to handle Picasso portraits too, there are no constraints on the location or size of the rectangles. How many possible faces (choice of its component rectangles) are there? In general, we only care about asymptotic complexity, so give your answer in the form of $O(n^c)$ or $O(c^n)$ for some integer c.

sol) For each component (eyes, ears, nose, or mouse) in a human face, we need to choose 4 independent parameters (x_0, y_0, h, w) to locate in the $n \times n$ image.

Therefore, for each component the complexity increased by $O(n^4)$. Since there are 6 components to locate, the asymptotic complexity is $O(n^24)$.

b. Suppose we have an $n \times n$ grid. We start in the upper-left corner (position $(1, 1)$), and we would like to reach the lower-right corner (position (n, n)) by taking single steps down and right. Define a function $c(i, j)$ to be the cost of touching position (i, j), and assume it takes constant time to compute. Note that $c(i, j)$ can be negative. Give an algorithm for computing the minimum cost in the most efficient way. What is the runtime (just give the big-O)?

sol) By filling in the $n \times n$ grid from upper-left corner while increasing x- or y- coordinate by 1 each time, we can calculate the minimum cost of each position.

Therefore filling the grid with total area n^2, we can compute the minimum cost in $O(n^2)$.

c. Suppose we have a staircase with n steps (we start on the ground, so we need n total steps to reach the top). We can take as many steps forward at a time, but we will never step backwards. How many ways are there to reach the top? Give your answer as a function of n.

For example: if $n = 3$, then the answer is 4. The four options are the following:

(i) take one step, take one step, take one step

(ii) take two steps, take one step

(iii) take one step, take two steps

(iv) take three steps.

sol) Let's denote the number of ways to reach the n staircase as $f(n)$. Since there exists only one way to reach the first step, $f(1) = 1 = 2^{1-1}$.

For 2 step case, we can take one step and one step again or take giant two steps. Hence, $f(2) = f(1) + 1 = 1 + 1 = 2 = 2^{2-1}$.

For $f(n)$, we can use all of previous ways. for instance, if we stepped k, then we can climb the rest $n - k$ steps in one giant step.

Therefore, we can formulate the recurrence relation as follows, $f(n) = \sum_{k=1}^{n-1} f(k) + 1$. And with mathematical approach, we can get $f(n) = 2^{n-1}$.

d. Consider the scalar-valued function $f(\mathbf{w})$ from Problem 1e. Devise a strategy that first does preprocessing in $O(nd^2)$ time, and then for any given vector $\mathbf{w}$, takes $O(d^2)$ time instead to compute $f(\mathbf{w})$.

Hint: Refactor the algebraic expression; this is a classic trick used in machine learning. Again, you may find it helpful to work out the scalar case first.

sol)

$$
\begin{aligned}
f(\mathbf{w}) &= \sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w})^2 + \lambda\sum_{m=1}^{d} w_m^2 \\
&= ((\mathbf{a}_1 - \mathbf{b}_1)^\mathsf{T}\mathbf{w})^2 + ((\mathbf{a}_1 - \mathbf{b}_2)^\mathsf{T}\mathbf{w})^2 + \cdots + ((\mathbf{a}_1 - \mathbf{b}_n)^\mathsf{T}\mathbf{w})^2 \\
&+ ((\mathbf{a}_2 - \mathbf{b}_1)^\mathsf{T}\mathbf{w})^2 + ((\mathbf{a}_2 - \mathbf{b}_2)^\mathsf{T}\mathbf{w})^2 + \cdots + ((\mathbf{a}_2 - \mathbf{b}_n)^\mathsf{T}\mathbf{w})^2 \\
&\vdots \\
&+ ((\mathbf{a}_n - \mathbf{b}_1)^\mathsf{T}\mathbf{w})^2 + ((\mathbf{a}_n - \mathbf{b}_2)^\mathsf{T}\mathbf{w})^2 + \cdots + ((\mathbf{a}_n - \mathbf{b}_n)^\mathsf{T}\mathbf{w})^2 \\
&+ \lambda \times (w_1^2 + w_2^2 + \cdots + w_1^2)
\end{aligned}
$$

Because each $(\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w}$ has $O(d)$ complexity, $\sum_{i=1}^{n}\sum_{j=1}^{n}((\mathbf{a}_i - \mathbf{b}_j)^\mathsf{T}\mathbf{w})^2$ is $O(nd^2)$ while $\lambda\sum_{m=1}^{d} w_m^2$ has complexity of $O(d)$.

Furthermore,

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

Since $\mathbf{a}_i\mathbf{a}_i^\mathsf{T}$, $\sum_{i=1}^{n}\mathbf{a}_i\sum_{j=1}^{n}\mathbf{b}_j^\mathsf{T}$, $\mathbf{b}_j\mathbf{b}_j^\mathsf{T}$ has $O(d^2)$ complexity, $f(\mathbf{w})$ can compute in $O(d^2)$.