Preparation for AI

Hwanjo Yu

Problem 1

Let $b_1, \ldots, b_n$ be real numbers representing positions on a number line. Let $w_1, \ldots, w_n$ be positive numbers representing the importance of these positions. Define the quadratic function: $f(x) = \frac{1}{2}\sum_{i=1}^{n} w_i(x-b_i)^2$. What value x minimizes $f(x)$? You can think about this problem as trying to find the point x that's not too far away from the points b_i's. Over time, hopefully you'll appreciate how nice quadratic functions are to minimize. What happens to this problem if some w_i's are negative?

Problem 2

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

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

Problem 3

Suppose you repeatedly roll a fair six-sided dice until the number of dots is 2 or fewer (and then you're done). Every time the dice turns up with a 4, you earn r points. What is the expected number of total points (as a function of r) that you will earn before you're done?

Problem 4

Suppose the probability of a coin turning up heads is $0 < p < 1$, and that we flip it 5 times and get $\{\mathrm{H}, \mathrm{T}, \mathrm{H}, \mathrm{T}, \mathrm{T}\}$. We know the probability (likelihood) of obtaining this sequence is $L(p) = p(1-p)p(1-p)(1-p) = p^2(1-p)^3$. Now let's go backwards and ask the question: what is the value of p that maximizes $L(p)$?

Hint: Consider taking the derivative of $\log L(p)$. Taking the derivative of $L(p)$ works too, but it is cleaner and more natural to differentiate $\log L(p)$. You can verify for yourself that the value of p which minimizes $\log L(p)$ must also minimize $L(p)$.

Problem 5

Let's practice taking gradients, which is a key operation for being able to optimize continuous functions. For $\mathbf{w} \in \mathbb{R}^d$ and constants $a_i, b_j \in \mathbb{R}^d$ and $\lambda \in \mathbb{R}$, define the scalar-valued function

$$f(\mathbf{w}) = \sum_{i=1}^{n} \sum_{j=1}^{n} (a_i^\top \mathbf{w} - b_j^\top \mathbf{w})^2 + \lambda \|\mathbf{w}\|_2^2,$$

where the vector is $\mathbf{w} = (w_1, \ldots, w_d)$ and $\|\mathbf{w}\|_2 = \sqrt{\sum_{j=1}^{d} w_j^2}$ 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:

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots \frac{\partial f(\mathbf{w})}{\partial w_d} \right).$$

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.

Problem 6

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, 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.

Problem 7

[3 points] Suppose we have n cities on the number line: $1, 2, \ldots, n$. Define a function $c(i, j)$ which returns the cost of going from city i to city j, and assume it takes constant time to compute. We want to travel from 1 to n via a set of intermediate cities, but only moving forwards. We can compute the minimum cost of doing this by defining the following recurrence: $f(j) = \min_{1 \le i < j} [c(i, j) + f(i)]$ for $j = 1, \ldots, n$. Give an algorithm for computing $f(n)$ for a fixed n in the most efficient way. What is the runtime (just give the big-O)?