Foundations - Stanford CS 221 Fall 2017-2018

Rohit Apte

2017-09-30

1 Problem 1: Optimization and probability

a $x_1, ..., x_n$ are real numbers representing positions on a number line. $w_1, ..., w_n$ are positive real numbers representing the importance of each of these positions. The quadratic function $f(\theta) = \frac{1}{2}\sum_{i=1}^n w_i(\theta - x_i)^2$. What value of θ minimizes $f(\theta)$?

Since f is a quadratic function, it is convex and has a global minimum. We can therefore find the value of θ that minimizes $f(\theta)$ by solving for $\frac{df(\theta)}{d\theta} = 0$.

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

Solving for the derivative = 0 we have

$$\begin{aligned}
\sum_{i=1}^n w_i(\theta - x_i) &= 0 \\
\Rightarrow \sum_{i=1}^n (w_i\theta - w_i x_i) &= 0 \\
\Rightarrow \sum_{i=1}^n w_i\theta - \sum_{i=1}^n w_i x_i &= 0 \\
\Rightarrow \theta\sum_{i=1}^n w_i - \sum_{i=1}^n w_i x_i &= 0 \\
\Rightarrow \theta &= \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}
\end{aligned}$$

If some of the $w_i s$ are negative (or all zero), we can have a situation where the denominator sums to zero which makes the solution blow up (infinity) or not be defined (if we get 0 divided by 0).

b Let $f(x) = \sum_{i=1}^{d} max_{s\in\{1,-1\}} sx_i$ and $g(x) = max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i$ where $x = (x_1, ..., x_d) \in \mathbb{R}^d$ is a real vector. Does $f(x) \leq g(x)$,$f(x) = g(x)$ or $f(x) \geq g(x)$?

$$f(x) = \sum_{i=1}^{d} max_{s\in\{1,-1\}} sx_i$$

$$\Rightarrow f(x) = \sum_{i=1}^{d} max(x_i, -x_i)$$

$$g(x) = max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i$$

$$\Rightarrow g(x) = max(\sum_{i=1}^{d} x_i, \sum_{i=1}^{d} -x_i)$$

We can reinterpret these as $f(x) = \sum_{i=1}^{d} |x|$ and $g(x) = |\sum_{i=1}^{d} x_i|$.

For two numbers a and b, we know that based on the triangle inequality $|a|+|b| \geq |a+b|$.
We can generalize this for more than 2 numbers.

Therefore $f(x) \geq g(x)$.

c Suppose you repeatedly roll a fair six-sided dice 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. What is the expected number of points (as a function of a and b) you will have when you stop?

Probability of rolling each number (1,2,3,4,5,6) = $\frac{1}{6}$
Therefore the expected number of points if we don't roll a 1 on the first roll is $\frac{-a}{6} + \frac{b}{6} = \frac{b-a}{6}$.
If we stop after n rolls, we cannot have rolled a 1 in the first $n-1$ rolls and have to roll a 1 at the n^{th} roll. The probability of that is $\frac{5}{6}^{n-1}\frac{1}{6}$
If we stop at the 4^{th} roll, we didn't roll a 1 in the first, second or third roll, and we rolled a 1 at the fourth roll. The expected number of points for that is given by $\frac{b-a}{6} + \frac{5}{6}\frac{b-a}{6} + \frac{5}{6}^2\frac{b-a}{6}$
For n rolls we have $\frac{b-a}{6} + \frac{5}{6}\frac{b-a}{6} + \frac{5}{6}^2\frac{b-a}{6} + \cdots + \frac{5}{6}^{n-2}\frac{b-a}{6}$

Therefore our expected number of points for infinite rolls (i.e. we never stop) is

$$\frac{b-a}{6} + \left(\frac{5}{6}\right)\frac{b-a}{6} + \left(\frac{5}{6}\right)^2\frac{b-a}{6} + \left(\frac{5}{6}\right)^3\frac{b-a}{6} + \left(\frac{5}{6}\right)^4\frac{b-a}{6}...$$
$$= \frac{b-a}{6} + (1 + \frac{5}{6} + \left(\frac{5}{6}\right)^2 + \left(\frac{5}{6}\right)^3 + \left(\frac{5}{6}\right)^4 + ...)$$

If we let $r = \frac{5}{6}$. When $r < 1$ the geometric series $(1 + r + r^2 + r^3 + ...)$ converges and we know that it converges to $\frac{1}{1-r} = 6$. Therefore the expected number of points for infinite rolls is $\frac{b-a}{6} \times 6 = b - a$.

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

Lets take the log of $L(p)$ since the value of p that maximizes $L(p)$ also maximizes $logL(p)$.

$$L(p) = p^4(1-p)^3$$
$$\therefore lnL(p) = 4ln(p) + 3ln(1-p)$$

The function is an inverted convex function so the maximum will be where the derivative is zero. Solving for this we get

$$4ln(p) + 3ln(1-p) = 0$$
$$\Rightarrow \frac{4}{p} + \frac{3}{1-p}(-1) = 0$$
$$\Rightarrow \frac{4}{p} + \frac{-3}{1-p} = 0$$
$$\Rightarrow \frac{4}{p} = \frac{3}{1-p}$$
$$\Rightarrow 4 - 4p = 3p$$
$$\Rightarrow p = \frac{4}{7}$$

The immediate interpretation is that it is not a fair coin. However, note that for 7 coin tosses we could never have a situation of equal heads and tails (since its an odd number of tosses). Based on the limited coin trials we calculate a value for p. We would have more confidence in this value if it was calculated over more coin tosses.

e For $w \in \mathbb{R}^d$ and constants $a_i, b_i \in \mathbb{R}^d$ and $\lambda \in \mathbb{R}$, define the scalar-valued function $f(w) = \sum_{i=1}^n \sum_{j=1}^n (a_i^T w - b_j^T w)^2 + \lambda \|w\|_2^2$. Compute the gradient $\nabla f(w)$ where the vector $w = (w_1, ..., w_d)^T \in \mathbb{R}_{d\times 1}$ $a_i, b_j \in \mathbb{R}_{d\times 1} \therefore a_i^T, b_j^T \in \mathbb{R}_{1\times d}$

For the scalar condition $w = w_1, a_i^T = a_i, b_j^T = b_j$ we have $f(w) = \sum_{i=1}^n \sum_{j=1}^n (a_i w - b_j w)^2 + \lambda w^2$.
Then $\nabla f(w) = \sum_{i=1}^n \sum_{j=1}^n (2(a_i w - b_j w)(a_i - b_j) + 2\lambda w = \sum_{i=1}^n \sum_{j=1}^n (2(a_i - b_j)w(a_i - b_j) + 2\lambda w$

In the vector case, we can re-factor the expression as $f(w) = \sum_{i=1}^n \sum_{j=1}^n (a_i^T w - b_j^T w)^2 + \lambda w^2 = \sum_{i=1}^n \sum_{j=1}^n ((a_i^T - b_j^T)w)^2 + \lambda \|w\|_2^2$
For d dimensions the partial derivatives are

$$\begin{aligned}
\frac{\partial f(w)}{\partial w_1} &= \sum_{i=1}^n \sum_{j=1}^n 2(a_i^T - b_j^T)w(a_i^T{}_1 - b_j^T{}_1) + 2\lambda w_1 \\
&= 2\sum_{i=1}^n \sum_{j=1}^n (a_i^T - b_j^T)w(a_i^T{}_1 - b_j^T{}_1) + 2\lambda w_1 \\
\frac{\partial f(w)}{\partial w_2} &= 2\sum_{i=1}^n \sum_{j=1}^n (a_i^T - b_j^T)w(a_i^T{}_2 - b_j^T{}_2) + 2\lambda w_2
\end{aligned}$$

therefore generally for any $k \in 1, \ldots d$

$$\frac{\partial f(w)}{\partial w_k} = 2\sum_{i=1}^n \sum_{j=1}^n (a_i^T - b_j^T)w(a_i^T{}_k - b_j^T{}_k) + 2\lambda w_k$$

2 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.

We have a total of 6 rectangles (2 eyes, 2 ears, 1 nose, 1 mouth). An algorithm to generate all possible combinations of ONE rectangle in an $n \times n$ grid would be as follows:

```
for topX=1 to n
    for topY=1 to n
        for bottomX=topX to n
            for bottomY=topY to n
                rectangle=(topX,topY,bottomX,bottomY)
```

This is $O(n^4)$. For a second rectangle, to generate ALL possible combinations of 2 rectangles, the code would be nested inside the first. So for 2 rectangles we would get $O(n^8)$. The third rectangle code would be $O(^(n^{12})$. And in general for n rectangles it would be $O(n^{4n})$ or $O(n^c)$ where $c = 4\times$ number of rectangles. Therefore for 6 rectangles it would be $O(n^{24})$.

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

Since we can only go down or right, if we find ourselves at a square (x, y) on the grid, then we either came from above it $(x - 1, y)$ or the left of it $(x, y - 1)$. Therefore the minimum cost of landing at square (x, y) can be defined as

$$minCost(x, y) = min(minCost(x - 1, y), minCost(x, y - 1)) + cost(x, y)$$

We now need to handle the edge cases. If we land on a square on the left most column, we can only have come from the square above it. Likewise if we landed on a square in the top row, we could only have come from a square to the left of it. Therefore we need to handle the cost for the special cases

$minCost(x, 1) = minCost(x - 1, 1) + cost(x, 1)$
and
$minCost(1, y) = minCost(1, y - 1) + cost(1, y)$

Since our x,y are pairs of integers going up to n, we can store the past calculated *minCost* into an array. The cost of lookup is $O(1)$.
Then we have the following algorithm

```
//assume minCost is an array of n x n initialized to zero
//Populate the cost for (1,1)
minCost(1,1)=cost(1,1)
//also fill in the edge cases
for i=2 to n:
      minCost(i,1)=minCost(i-1,1)+cost(i,1)
      minCost(1,i)=minCost(1,i-1)+cost(1,i)

for i=2 to n:
      for j=2 to n:
            minCost(i,j)=min(minCost(i-1,1),minCost(i,j-1))+cost(i,j)
```

The nested for loops give us complexity of $O(n^2)$.

c Suppose we have a staircase with steps (we start on the ground, so we need 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: (1) take one step, take one step, take one step (2) take two steps, take one step (3) take one step, take two steps (4) take three steps.

We are effectively trying to see how to write n as a sum of other numbers. Recall that a composition of n is a way or writing n as the sum of a sequence of (strictly) positive integers.
The number of compositions of n into k parts is
$\binom{n-1}{k-1}$
Therefore for all possible combinations of steps we have
$\sum_{k=1}^{n} \binom{n-1}{k-1} = 2^{n-1}$.
Therefore there are 2^{n-1} ways to reach the top of the stairs (assuming $n \geq 1$).

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

$a_i, b_j \in \mathbb{R}_{d\times 1} \therefore a_i^T, b_j^T \in \mathbb{R}_{1\times d}$
$f(w) = \sum_{i=1}^n \sum_{j=1}^n (a_i^T w - b_j^T w)^2 + \lambda \|w\|_2^2$
We can expand the term we need to square as follows
$f(w) = \sum_{i=1}^n \sum_{j=1}^n ((a_i^T w)^2 + (b_j^T w)^2 - 2(a_i^T w b_j^T w)) + \lambda \|w\|_2^2$
Lets look at each term
$(a_i^T w)^2 = (a_i^T w)(a_i^T w) = (w^T a_i)(a_i^T w) = w^T(a_i a_i^T)w$
$(b_j^T w)^2 = (b_j^T w)(b_j^T w) = (w^T b_j)(b_j^T w) = w^T(b_j b_j^T)w$
$2(a_i^T w b_j^T w) = 2(w^T a_i b_j^T w) = 2w^T(a_i b_j^T)w$
We can preprocess $a_i^T a_i$, $b_j^T b_j$ and $a_i^T b_j$. Each of these is $O(d^2)$.
For the $\sum_{i=1}^n \sum_{j=1}^n ((a_i^T w)^2 + (b_j^T w)^2 - 2(a_i^T w b_j^T w))$ we can rewrite this as
$n\sum_{i=1}^n w^T(a_i a_i^T)w + n\sum_{j=1}^n w^T(b_j b_j^T)w - 2\sum_{i=1}^n \sum_{j=1}^n w^T(a_i b_j^T)w$
The first 2 terms are $O(nd^2)$. The third term is $O(n^2 d^2)$. Lets see if we can improve on it.
$\sum_{i=1}^n \sum_{j=1}^n (a_i b_j^T) = (\sum_{i=1}^n a_i)\left(\sum_{j=1}^n b_j^T\right)$
We transform the double summation to single summation and therefore go from $O(n^2)$ to $O(n)$ on the summation. The matrix multiplication is $O(d^2)$.
The total preprocessing cost is therefore $O(nd^2)$.

For the norm component we have
$\|w\|_2^2 = \sum_{k=1}^d w_k^2 = w_1^2 + \cdots + w_d^2 = w^T w$
Once the preprocessing us done we have $a^T a$ which is 1×1, i.e. a scalar. Similarly $b^T b$ is a scalar. So we can refactor $w^T(a_i a_i^T)w$ as $w^T(C^T)w = Cw^T w$ where C is the scalar $a^T a$. Similarly for $w^T(b_j b_j^T)w$. And the vector norm component is $w^T w$. This is $O(d^2)$
Lastly, $a_i b_j^T$ is $d \times d$ so $w^T(a_i b_j^T)w$ is also $O(d^2)$.
Therefore we have the time to compute w as $O(d^2)$.