Homework 1: Foundations

Brandon McKinzie

1. Problem 1: Optimization and Probability

(a). *Let $x_1, \dots x_n$ be real numbers representing positions on a number line. Let $w_1, \dots w_n$ be (strictly) 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 $f(\theta)$? 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. How will your answer change if some of the w_i's are negative?*

We can find $\theta^* = \arg\min_\theta f(\theta)$ by taking the derivative of f with respect to θ, setting to zero, and solving for θ:

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

$$0 = \frac{1}{2}\sum_{i=1}^{n} w_i \cdot 2 \cdot (\theta - x_i) \tag{2}$$

$$0 = \sum_{i=1}^{n} w_i(\theta - x_i) \tag{3}$$

$$0 = \theta\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i x_i \tag{4}$$

$$\theta = \frac{\boldsymbol{w} \cdot \boldsymbol{x}}{\sum_{i=1}^{n} w_i} \tag{5}$$

And we see that θ^* is the weighted average of the x_i's, where the weights are the w_i's. One could also interpret it as $\theta^* = \mathbb{E}[x]$ with each $x \sim p(x)$ and $p(x_i) = w_i / \sum_j w_j$ (the empirical distribution). If some of the w_i values are negative, then we can no longer interpret it as a weighted average or expectation value. It would also drive θ further from any x_i with $w_i < 0$ instead of closer.

(b). *In this class, there will be a lot of sums and maxes. Let's see what happens if we switch the order. Let*

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} \max_{s \in \{1,-1\}} s x_i \tag{6}$$

$$g(\boldsymbol{x}) = \max_{s \in \{1,-1\}} \sum_{i=1}^{d} s x_i \tag{7}$$

where $\boldsymbol{x} \in \mathbb{R}^d$. Does $f(\boldsymbol{x}) \leq g(\boldsymbol{x})$, $f(\boldsymbol{x}) = g(\boldsymbol{x})$, $f(\boldsymbol{x}) \geq g(\boldsymbol{x})$ hold for all $\boldsymbol{x}$? Prove it.

Let $\mathcal{P} = \{p \in [1..d] \mid x_p \geq 0\}$. Similarly, let $\mathcal{N} = \{n \in [1..d] \mid x_n < 0\}$. Then

$$\sum_{i=1}^{d} x_i = \sum_{p \in \mathcal{P}} |x_p| - \sum_{n \in \mathcal{N}} |x_n| \tag{8}$$

Also note that $|x_i| \geq x_i$ for all $x_i \in \mathbb{R}$.

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} \max(x_i, -x_i) = \sum_{i=1}^{d} |x_i| \tag{9}$$

$$g(\boldsymbol{x}) = \max \left[\sum_{i=1}^{d} x_i, -\sum_{i=1}^{d} x_i \right] \tag{10}$$

$$= \max \left[\sum_{p \in \mathcal{P}} |x_p| - \sum_{n \in \mathcal{N}} |x_n|, \sum_{n \in \mathcal{N}} |x_n| - \sum_{p \in \mathcal{P}} |x_p| \right] \tag{11}$$

Since it is true that both

$$\sum_{i=1}^{d} |x_i| \geq \sum_{p \in \mathcal{P}} |x_p| - \sum_{n \in \mathcal{N}} |x_n| \tag{12}$$

$$\text{and} \quad \sum_{i=1}^{d} |x_i| \geq \sum_{n \in \mathcal{N}} |x_n| - \sum_{p \in \mathcal{P}} |x_p| \tag{13}$$

we see that $f(\boldsymbol{x}) \geq g(\boldsymbol{x})$ for all $\boldsymbol{x}$.

(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?*

Let random variable $X_i : [1..6] \mapsto \mathbb{R}$ map the outcome of roll i to the number of points received on that turn only (note that we are not told a or b are integers, so I use $\mathbb{R}$ to maintain generality):

$$\mathbb{E}[X_i] = \sum_{j=1}^{6} X_i(j)P(j) \tag{14}$$

$$= \frac{1}{6}\sum_{j=1}^{6} X_i(j) \tag{15}$$

$$= \frac{b-a}{6} \tag{16}$$

which gives us the expected number of points for any given roll. Now, let N denote the random variable representing the total number of rolls we get for a given trial run (with our first 1 occurring on the Nth roll). We can use the law of total expectation along with linearity of expectation to compute the expected number of points $X = X_1 + \ldots + X_N$ for a given trial.

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid N{=}n]] = \sum_{n=1}^{\infty} \mathbb{E}[X \mid N{=}n]\, P(N{=}n) \tag{17}$$

$$\mathbb{E}[X \mid N{=}n] = \sum_{i=1}^{n-1} \mathbb{E}[X_i \mid N{=}n] \tag{18}$$

$$= (n-1)\frac{b-a}{5} \tag{19}$$

$$\therefore \mathbb{E}[X] = \sum_{n=1}^{\infty} (n-1)\frac{b-a}{5}(5/6)^{n-1}(1/6) \tag{20}$$

$$= \sum_{n=0}^{\infty} n\frac{b-a}{5}(5/6)^{n}(1/6) \tag{21}$$

$$= \frac{b-a}{5}(1/6)(5/6)36 \tag{22}$$

$$= b - a \tag{23}$$

(d). *You are playing a game with a fair five-sided die (sides 1,2,3,4,5 and probability $\frac{1}{5}$ of coming up each of these). At each turn, you have an option to quit and win 15 points (and thus roll no more) or to win 3 points and roll the dice. If at any point, you roll an even number, the game ends, and you leave with your total winnings. Else you conitnue onto the next turn. What is the optimal strategy and why? What would change if the dice was weighted so that the probability of rolling an even number was 0?*

The probability of continuing the game when you roll is $P(odd) = 3/5$, and the probability of the game ending on a given roll is $P(even) = 2/5$. The random variable X denotes the number of odd roles until the first even[1]. This follows a geometric distribution:

$$P(X{=}k) = p_{odd}^{k} \cdot p_{even} \tag{24}$$

$$\mathbb{E}[X] = \frac{p_{odd}}{p_{even}} = 3/2 \tag{25}$$

Let Y_n denote the total number of points received if we commit to exactly n rolls and then quit. We can find a formula for $\mathbb{E}[Y_n]$ as follows.

$$\mathbb{E}[Y_1] = 3 \cdot \frac{3}{5} + 0 \cdot \frac{2}{5} \tag{26}$$

$$\mathbb{E}[Y_2] = 6 \cdot \frac{3}{5}^2 + 0 \cdot \frac{2}{5}\frac{3}{5} + 0 \cdot 1 \cdot \frac{2}{5} \tag{27}$$

$$\mathbb{E}[Y_n] = (3n) \cdot \frac{3}{5}^n \tag{28}$$

The optimal strategy would be to commit to rolling $n^* = \arg\max_n \mathbb{E}[Y_n]$ times. Of course, this yields a non-integer value of n, so we choose from the 2 integers on either side with larger value of $\mathbb{E}[Y_n]$. Taking the derivative, and solving for n yields $n = 1/\ln(5/3) \approx 1.96$. Since $\mathbb{E}[Y_2] > \mathbb{E}[Y_1]$, the optimal strategy is to (attempt to) roll twice and then quit, since that corresponds to the maximal expected return.

This is, of course, assuming a hyper-rational agent with zero risk aversion. If, for example, we got 1 million dollars if we quit instead of roll, any logical human will not choose to roll at all, and just take the 1 million dollars.

If $P(even) = 0$, then you should continue playing indefinitely, since each roll guarantees an additional 3 points.

[1] So $X = k$ means k odd roles in a row, and with the result of the k+1 roll as even.

(e). *What value of p maximizes $L(p)$? What is an intuitive interpretation of this value of p?*

$$\log L(p) = 4\log(p) + 3\log(1-p) \tag{29}$$

$$\frac{\partial \log L(p)}{\partial p} = \frac{4}{p} - \frac{3}{1-p} \tag{30}$$

$$\frac{4}{p^*} = \frac{3}{1-p^*} \tag{31}$$

$$3p^* = 4(1-p^*) \tag{32}$$

$$p^* = \frac{4}{7} \tag{33}$$

Intuitively, we maximize the probability of observing 4 heads in 7 tosses if we set the probability of getting heads to 4/7, which also corresponds to $\mathbb{E}[X_7] = 4$ (X_7 is the random variable for number of heads in 7 tosses).

(f). *Let's practice taking gradients, which is a key operation for being able to optimize continuous functions. Compute the gradient* $\nabla f(\boldsymbol{w})$*, where*

$$f(\boldsymbol{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})^2 + \lambda||\boldsymbol{w}||_2^2 \tag{34}$$

I'll be using the following property:

$$\nabla_{\boldsymbol{w}}\boldsymbol{x}^T\boldsymbol{w} = \boldsymbol{x} \tag{35}$$

We can then compute the gradient as follows:

$$\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\nabla_{\boldsymbol{w}}(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})^2 + \lambda\nabla_{\boldsymbol{w}}||\boldsymbol{w}||_2^2 \tag{36}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}2(\boldsymbol{a}_i^T\boldsymbol{w} - \boldsymbol{b}_j^T\boldsymbol{w})(\boldsymbol{a}_i - \boldsymbol{b}_j) + 2\lambda\boldsymbol{w} \tag{37}$$

$$\tag{38}$$

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.*

Starting with the simplest setting of a single rectangular component. There are exactly $\sum_{w=1}^{n} \sum_{h=1}^{n} (n - w + 1)(n - h + 1)$ possibilities, which is $\mathcal{O}(n^4)$ overall. Since the rectangles are allowed to overlap[2], the fact that there are 6 squares instead of 1 only introduces a constant factor, keeping the overall complexity still at $\mathcal{O}(n^4)$.

[2] I was explicitly told this during the SCPD office hour

(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)?*

Let $C^*(i, j)$ denote the minimum total cost to reach (i, j) from the starting point. The final minimum cost, $C^*(n, n)$, is a result of coming from one of two positions on the previous step, either $(n-1, n)$ or $(n, n-1)$. Therefore,

$$C^*(n, n) = min\left[C^*(n-1, n) + c(n, n), C^*(n, n-1) + c(n, n)\right] \tag{39}$$

In other words, all we have to do is walk along each possible path while caching the values of $C^*(i, j)$ as we go. Due to the black-box definition of $c(i, j)$ that we've been given, we must visit all possible nodes, giving us a lower-bound of $\mathcal{O}(n^2)$. Fortunately, since we are caching our results, and since the total minimum cost to reach a given node can be computed in terms of the total minimum cost of previous nodes, we only have to compute $C^*(i, j)$ once for any given (i, j) and can do each computation in $\mathcal{O}(1)$ time due to our caching approach. This results in a total runtime of $\mathcal{O}(n^2)$ which means it is maximally efficient.

(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: (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.*

Let $X(n)$ denote the number of ways to reach the top if there are n stairs. We can observe a pattern by writing out the first few cases of n.

$$X(1) = 1 \tag{40}$$
$$X(2) = 2 \tag{41}$$
$$X(3) = 1 + 2 + 1 = 4 \tag{42}$$
$$X(4) = 1 + 4 + 2 + 1 = 8 \tag{43}$$
$$X(5) = 1 + 8 + 4 + 2 + 1 = 16 \tag{44}$$

The recursion is now clear. We can then progressively simplify to get the final function of n as follows.

$$X(n) = 1 + \sum_{i=2}^{n} 2^{i-2} \tag{45}$$
$$= 1 + \sum_{i=1}^{n-1} 2^{i-1} \tag{46}$$
$$= 2^{n-1} \tag{47}$$

(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 w, takes $O(d^2)$ time instead to compute f(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.*

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w}-\boldsymbol{b}_j^T\boldsymbol{w})^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(\boldsymbol{a}_i^T\boldsymbol{w})^2+(\boldsymbol{b}_j^T\boldsymbol{w})^2-2(\boldsymbol{a}_i^T\boldsymbol{w})(\boldsymbol{b}_j^T\boldsymbol{w}) \tag{48}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{w}^T\boldsymbol{a}_i\boldsymbol{a}_i^T\boldsymbol{w}+\boldsymbol{w}^T\boldsymbol{b}_j\boldsymbol{b}_j^T\boldsymbol{w}-2\boldsymbol{w}^T\boldsymbol{a}_i\boldsymbol{b}_j^T\boldsymbol{w} \tag{49}$$

$$= \boldsymbol{w}^T\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\boldsymbol{a}_i\boldsymbol{a}_i^T+\boldsymbol{b}_j\boldsymbol{b}_j^T-2\boldsymbol{a}_i\boldsymbol{b}_j^T\right]\boldsymbol{w} \tag{50}$$

$$= \boldsymbol{w}^T\left[\left(\sum_{i=1}^{n}\boldsymbol{a}_i\boldsymbol{a}_i^T\right)+\left(\sum_{j=1}^{n}\boldsymbol{b}_j\boldsymbol{b}_j^T\right)-2\left(\sum_{i=1}^{n}\boldsymbol{a}_i\right)\left(\sum_{j=1}^{n}\boldsymbol{b}_j^T\right)\right]\boldsymbol{w} \tag{51}$$

$$\tag{52}$$

The preprocessing is shown in the innermost parentheses. The computation of $\boldsymbol{a}_i\boldsymbol{a}_i^T$ is $\mathcal{O}(d^2)$ and it is computed n times, for an overall $\mathcal{O}(nd^2)$. The same is true for the $\boldsymbol{b}_j$ parentheses. Similarly, the individual sums over the $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ are each $\mathcal{O}(n)$, and then incur the additional d^2 when computing the resultant $d \times d$ matrix, for again a total of $\mathcal{O}(nd^2)$. Therefore, our preprocessing is $\mathcal{O}(3nd^2) \equiv \mathcal{O}(nd^2)$. Our preprocessing yields a $d \times d$ matrix $\boldsymbol{D}$, and for any vector $\boldsymbol{w}$ we simply compute $\boldsymbol{w}^T\boldsymbol{D}\boldsymbol{w}$, which is $\mathcal{O}(d^2)$.