Homework 0 – DSC 240
Winter 2026
Diana Freeman: A16574234

Problem 1

(a)

Find theta that minimizes $f(\theta)$

First start by Expanding

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

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

$$= \sum_{i=1}^{n} w_i x_i^2 - 2\theta \sum_{i=1}^{n} w_i x_i + \theta^2 \sum_{i=1}^{n} w_i$$

Differentiate with respect to θ:

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

Set derivative equal to zero:

$$-2 \sum_{i=1}^{n} w_i x_i + 2\theta \sum_{i=1}^{n} w_i = 0$$

$$\theta \sum_{i=1}^{n} w_i = \sum_{i=1}^{n} w_i x_i$$

$$\theta^* = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

If some $w_i < 0$, the function may not be convex and the minimum may not exist. The function may be unbounded below.

(b)

Probability that the two tallest kids are in the same group: Fix the two tallest kids. Place the tallest child into a group. There are $2n - 1$ remaining positions for the second tallest child. There are $n - 1$ spots in the same group and n spots in the other group.

Giving:

$$\frac{n-1}{2n-1}$$

Probability that the two tallest kids are in different groups: Using the same logic as above by fixing the tallest kid and counting available positions, since its binary then

$$\frac{n}{2n-1}$$

(c)

Let K be the event that the candidate knows the answer and C be the event that the answer is correct.

$$P(C \mid K) = 0.99, \quad P(C \mid G) = \frac{1}{k}$$
$$P(K) = p, \quad P(G) = 1 - p$$

Using Bayes' rule:

$$P(K \mid C) = \frac{P(C \mid K)P(K)}{P(C \mid K)P(K) + P(C \mid G)P(G)}$$
$$= \frac{0.99p}{0.99p + \frac{1}{k}(1-p)}$$

(d)

The likelihood of observing the sequence is:

$$L(p) = p^6(1-p)^4$$

Take the log-likelihood:

$$\log L(p) = 6 \log p + 4 \log(1-p)$$

Differentiate:

$$\frac{d}{dp} \log L(p) = \frac{6}{p} - \frac{4}{1-p}$$

Set equal to zero:

$$\frac{6}{p} = \frac{4}{1-p}$$
$$6(1-p) = 4p$$
$$6 = 10p$$
$$p = 0.6$$

(e)

Let $w \in \mathbb{R}^d$ and $x_i \in \mathbb{R}^d$ (column vectors), $y_i \in \mathbb{R}$, and $\lambda \geq 0$. Define

$$F(w) = \sum_{i=1}^{n} (x_i^T w - y_i)^2 + \lambda \sum_{j=1}^{d} w_j^2 = \sum_{i=1}^{n} (x_i^T w - y_i)^2 + \lambda \|w\|_2^2.$$

Write $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$ and $w = (w_1, \ldots, w_d)^T$, so that

$$x_i^T w = \sum_{k=1}^{d} x_{ik} w_k.$$

Fix an index $j \in \{1, \ldots, d\}$. Using the chain rule,

$$\frac{\partial}{\partial w_j}(x_i^T w - y_i)^2 = 2(x_i^T w - y_i) \cdot \frac{\partial}{\partial w_j}(x_i^T w - y_i).$$

But $\frac{\partial}{\partial w_j}(x_i^T w - y_i) = \frac{\partial}{\partial w_j}\Big(\sum_{k=1}^{d} x_{ik} w_k - y_i\Big) = x_{ij}$, so

$$\frac{\partial}{\partial w_j}(x_i^T w - y_i)^2 = 2(x_i^T w - y_i)\, x_{ij}.$$

Therefore,

$$\frac{\partial}{\partial w_j} \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \sum_{i=1}^{n} 2(x_i^T w - y_i)\, x_{ij}.$$

For the regularization term,

$$\frac{\partial}{\partial w_j}\Big(\lambda \sum_{k=1}^{d} w_k^2\Big) = \lambda \sum_{k=1}^{d} \frac{\partial}{\partial w_j}(w_k^2) = \lambda \cdot 2w_j.$$

Putting the two pieces together,

$$\frac{\partial F(w)}{\partial w_j} = 2 \sum_{i=1}^{n} (x_i^T w - y_i)\, x_{ij} + 2\lambda w_j.$$

Now stack these partial derivatives into the gradient vector:

$$\nabla F(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \vdots \\ \frac{\partial F}{\partial w_d} \end{bmatrix} = 2 \sum_{i=1}^{n} (x_i^T w - y_i)\, x_i + 2\lambda w.$$

(f)

Let $x = (x_1, \ldots, x_n)^T$ and define

$$f(x_1, \ldots, x_n) = \log\Big(\sum_{j=1}^{n} e^{x_j}\Big).$$

Introduce an intermediate variable

$$S(x) := \sum_{j=1}^{n} e^{x_j},$$

so that

$$f(x) = \log(S(x)).$$

Fix an index $i \in \{1, \ldots, n\}$. By the chain rule,

$$\frac{\partial f}{\partial x_i} = \frac{d}{dS}\big(\log S\big)\Big|_{S=S(x)} \cdot \frac{\partial S}{\partial x_i}.$$

First term:

$$\frac{d}{dS}(\log S) = \frac{1}{S} \quad \Longrightarrow \quad \frac{d}{dS}\big(\log S\big)\Big|_{S=S(x)} = \frac{1}{S(x)}.$$

Second term:

$$\frac{\partial S}{\partial x_i} = \frac{\partial}{\partial x_i}\Big(\sum_{j=1}^{n} e^{x_j}\Big) = \sum_{j=1}^{n} \frac{\partial}{\partial x_i}\big(e^{x_j}\big).$$

Now note that

$$\frac{\partial}{\partial x_i}\left(e^{x_j}\right) = \begin{cases} e^{x_i}, & j = i, \\ 0, & j \neq i, \end{cases}$$

so the sum collapses to

$$\frac{\partial S}{\partial x_i} = e^{x_i}.$$

Putting the two pieces together,

$$\frac{\partial f}{\partial x_i} = \frac{1}{S(x)} \cdot e^{x_i} = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}.$$

Therefore, the gradient vector is

$$\nabla f(x) = \begin{bmatrix} \frac{e^{x_1}}{\sum_{j=1}^{n} e^{x_j}} \\ \vdots \\ \frac{e^{x_n}}{\sum_{j=1}^{n} e^{x_j}} \end{bmatrix}.$$

(g)

Let

$$f(x_1, \ldots, x_n) = \log \sum_{i=1}^{n} e^{x_i}.$$

Let $M = \max_{1 \leq i \leq n} x_i$. Then for every i, we have $x_i \leq M$, so $e^{x_i} \leq e^M$.

Lower bound. Choose an index $i^\star$ such that $x_{i^\star} = M$ (it exists by definition of max). Then

$$\sum_{i=1}^{n} e^{x_i} \geq e^{x_{i^\star}} = e^M.$$

Taking log on both sides (log is increasing), we get

$$\log \sum_{i=1}^{n} e^{x_i} \geq \log(e^M) = M.$$

So

$$M \leq f(x_1, \ldots, x_n).$$

Upper bound. Since for every i we have $e^{x_i} \leq e^M$, summing over $i = 1, \ldots, n$ gives

$$\sum_{i=1}^{n} e^{x_i} \leq \sum_{i=1}^{n} e^M = n e^M.$$

Taking log on both sides (log is increasing), we get

$$\log \sum_{i=1}^{n} e^{x_i} \leq \log(n e^M) = \log n + \log(e^M) = \log n + M.$$

So

$$f(x_1, \ldots, x_n) \leq M + \log n.$$

Conclusion. Combining the two bounds,

$$\max_i x_i = M \leq \log \sum_{i=1}^{n} e^{x_i} \leq M + \log n = \max_i x_i + \log n.$$

Problem 2

(a)

```
import numpy as np

A = np.arange(1,21).reshape(5,4)
B = np.arange(1,13).reshape(4,3)
A @ B
```

Output:

```
[[ 70  80  90]
 [158 184 210]
 [246 288 330]
 [334 392 450]
 [422 496 570]]
```

(b)

Dense matrix-vector multiplication has time complexity $O(n^2)$. If the matrix is sparse with $nnz(A)$ non-zero elements, the time complexity is $O(nnz(A))$.

```
from scipy.sparse import diags

def sparse_matrix(n):
    return diags([1,-1],[0,1],shape=(n-1,n))
```

Output:

```
<Compressed Sparse Row sparse matrix of dtype 'float64'>
```

(c)

```
def word_count(filename):
    counts = {}
    with open(filename) as f:
        for line in f:
            for word in line.split():
                counts[word] = counts.get(word,0)+1
    return counts
```

I used a dictionary to store word frequencies, where keys are words and values are counts. The file is read line by line, each line is split into words, and the dictionary is updated for each word. The time complexity is O(N), where N is the total number of words in the file. The space complexity is O(U), where U is the number of unique words.

(d)

```
memo = {0:0, 1:1}

def fibonacci(n):
    if n not in memo:
        memo[n] = fibonacci(n-1) + fibonacci(n-2)
    return memo[n]
```

Using recursion with bookkeeping (memoization), each Fibonacci number up to n is computed at most once and stored in a dictionary. Therefore, the time complexity is O(n), and the space complexity is also O(n).