XCS221 Assignment 0 — Foundations

THIS ASSIGNMENT IS COMPLETELY OPTIONAL.
YOU WILL RECEIVE NO CREDIT FOR ANY PART OF THIS ASSIGNMENT.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These unit tests will verify only that your code runs without errors on obvious test cases. These tests so not require the instructor solution code and can therefore be run on your local computer.
- `hidden`: These unit tests will verify that your code produces correct results on complex inputs and tricky corner cases. Since these tests require the instructor solution code to verify results, only the setup and inputs are provided. When you run the autograder locally, these test cases will run, but the results will not be verified by the autograder. When your run the autograder online, these tests will run and you will receive feedback on any errors that might occur.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Introduction

Welcome to your first XCS221 assignment! The goal of this assignment is to sharpen your math and programming skills needed for this class. If you meet the prerequisites, you should find these problems relatively innocuous. Some of these problems will occur again as subproblems of later homeworks, so make sure you know how to do them.

1. **Optimization and Probability**

In this class, we will cast a lot of AI problems as optimization problems, that is, finding the best solution in a rigorous mathematical sense. At the same time, we must be adroit at coping with uncertainty in the world, and for that, we appeal to tools from probability.

(a) **[1 point] Optimization**

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.

(b) **[1 point] Optimization**

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 $\mathbf{x}$? Please provide a formal proof explaining your answer.

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

(c) **[1 point] Probability**

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: We recommend defining a recurrence.

(d) **[1 point] Probability**

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 $L(p)$ (you are not required to prove this in your solution).*

(e) **[1 point] Linear Algebra**

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

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

where the vector is $\mathbf{w} = (w_1, \ldots, w_d)^\top$ 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:

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

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.

2. **Complexity**

When designing algorithms, it's useful to be able to do quick back of the envelope calculations to see how much time or space an algorithm needs. Hopefully, you'll start to get more intuition for this by being exposed to different types of problems.

(a) **[1 point] Complexity**

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.

(b) **[1 point] Dynamic Programming**

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)?

(c) **[1 point] Recursion**

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.

(d) **[1 point] Complexity**

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.

3. **Optimization and Probability**

In this problem, you will implement a bunch of short functions. The main purpose of this exercise is to familiarize yourself with Python, but as a bonus, the functions that you will implement will come in handy in subsequent homeworks.

If you are new to Python, the following provide pointers to various tutorials and examples for the language:

- Python for Programmers
- Example programs of increasing complexity

(a) **[3 points]**

Implement `findAlphabeticallyLastWord` in `submission.py`.

(b) **[2 points]**

Implement `euclideanDistance` in `submission.py`.

(c) **[3 points]**

Implement `mutateSentences` in `submission.py`.

(d) **[2 points]**

Implement `sparseVectorDotProduct` in `submission.py`.

(e) **[2 points]**

Implement `incrementSparseVector` in `submission.py`.

(f) **[3 points]**

Implement `findSingletonWords` in `submission.py`.

(g) **[5 points]**

Implement `computeLongestPalindromeLength` in `submission.py`.

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

1.a

1.b

1.c

1.d

1.e

2.a

2.b

2.c

2.d