Homework 10
Due Friday, May 1 at 5:00 PM

"I don’t think you’re human if you don’t get nervous.” - Sidney Crosby, gold-medal hockey player

Handing In

To hand in a homework, go to the directory where your work is saved and run `cs0160_handin hwX` where X is the number of the homework. Make sure that your written work is saved as a .pdf file, and any Python problems are completed in the same directory or a subdirectory. You can re-handin any work by running the handin script again. We’ll only grade your most recent submission. To install stencil Python files for a homework, run `cs0160_install hwX`. You will lose points if you do not hand in your written work as a .pdf file.

1 Written Problems

Problem 10.1

Decision Trees

1. Explain the difference between training data and testing data in the context of a decision tree.

2. Define: entropy, remainder, and information gain (in the context of a decision tree). What does having a high entropy mean? How are these concepts related?

3. The ID3 algorithm we learned in class is a recursive algorithm. Walk through each of the four recursive cases in this algorithm, specifying what to pass into the recursive call/what to return.
Problem 10.2

Racing

Ignorant Bet Person (IBP) is watching 2 track athletes (A and B) race against each other with a group of experts - Jesse Owens (J), Caster Semenya (C), Michelle Carter (M), and the Valerie Adams (V). True to the name, IBP wants to bet on which athlete will win each race, but sadly knows nothing about the athletes that are racing. IBP instead decides to bet based on the advice that the experts provide.

Ideally, in an offline situation, IBP would only listen to the expert who knows the most about track. However, and once more true to the name, IBP doesn’t know who that is! Instead, to deal with this online situation, IBP decides to bet using a weighted majority vote based on the advice given by the experts. IBP initially considers each of the other members equally with respective weights of 1. When an expert gives bad advice, IBP weighs all of their subsequent advice exactly one third of the previous weight. If there is a tie, IBP will always bet on athlete A winning the race. Given the table below of advice and actual winner, fill in IBP’s bet for each race. The first bet has been filled out for you.

How much weight do each of the experts have after the end of the fourth race?

<table>
<thead>
<tr>
<th>Race</th>
<th>J</th>
<th>C</th>
<th>M</th>
<th>V</th>
<th>IBP</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>
Problem 10.3

Computability

In class, we talked about classifications of computability (P, NP, NP-Hard, NP-Complete).

Part 1:

For the following problems, state which complexity class each of the problem falls under, and explain your answer in one to two sentences. For problem 3, in three or four sentences, describe the algorithm to solve this problem.

1. Sorting a list of numbers
2. Creating a seating arrangement of $n$ people at $k$ tables, such that no two people who hate each other are seated at the same table. You can assume that you know how two arbitrary people $k_A$ and $k_B$ feel about each other.
3. Determining whether it’s possible to assign each of a graph’s vertices one of two colors, such that no two vertices that share an edge have the same color.

Part 2:

In class, we talked about P and NP as if they were known to be different complexity classes. In reality, it’s not known whether P = NP, although in most cases we assume P $\neq$ NP.


Consider that RSA encryption (one of the most common encryption techniques) is in NP. In one to two sentences, explain what it would mean with regards to this if it was proven that P = NP.
Problem 10.4

Machine Learning

1. Consider the following two applications for a neural network and state how large the final layer should be:

   (a) Given an input vector of numbers, we want to return a True or False classification.

   (b) Given an input black and white image of a handwritten digit, we can convert this image to a 1D vector of 1s and 0s corresponding to whether each pixel is black or white. We want to find the digit (0-9) represented in the image given this input 1D vector.

2. Given a list of input values, describe how a perceptron would calculate an output value.
Problem 10.5

Algorithmic Fairness

We learned about machine learning (ML) and how ML models can be biased in class. We will cover this in more depth in class on Tuesday (4/28).

Two big challenges in machine learning right now are: (1) to better understand what it means for ML algorithms to be biased; and (2) to design ML algorithms that are fair. This sub-field of research is often called “algorithmic fairness.” There are currently over 20 different ways to define fairness for algorithms and computer scientists have shown that some of these notions are related and some are contradictory.

Think about a setting where ML algorithms could be deployed (for example, self-driving cars, spam detection, computer vision, robotics, language translation) and describe ways in which these algorithms could behave in an unfair/biased manner. How would you test such an algorithm to determine if it is biased? How would you fix it?

Be creative! We are not looking for an ideal solution, just some thought and effort.
2 Python Problems

Problem 10.6

Functional Programming Practice

In class we talked about two higher order functions called map and reduce. Solve each of the following problems using only python’s built in map or reduce (it is up to you to decide which one is appropriate for the problem). The functions you pass into map or reduce must be anonymous functions. Your solutions should go in the stencil functional.py. You might have to think about each of these for a while, but the solution to each one is a single line of code (besides the error handling).

Part 1: apply_all

Fill in the function apply_all in the stencil. This function takes in a list of unary (one argument) functions and a number. It returns a new list with each of those functions applied to that number. In other words, if you pass in the list of functions \([f(x), g(x), h(x)]\) and the number \(n\), apply_all should produce \([f(n), g(n), h(n)]\).

Example

- apply_all([λ x: x+1, λ x: x+2, λ x: x+3], 4) → [5, 6, 7]

Part 2: compose

Fill in the function compose in the stencil. This function takes in a list of unary functions and a number \(n\). Compose should compose all of the functions in the list and apply to \(n\) to produce a single number. The inner-most function in the composition will be the first function in the input list, and the outer-most function in the composition will be the last function in the input list. In other words, if you pass in the list of functions \([f(x), g(x), h(x)]\) and the number \(n\), compose should produce \(h(g(f(n)))\).

Example

- compose([λ x: x+1, λ x: x+2, λ x: x+3], 4) → 10.

Part 3: list_compose_steps

Fill in list_compose_steps, which should take in a list of unary functions and a number \(n\). It should return a list with each of the intermediate values produced by compose. In other words, if you pass in the list of functions \([f(x), g(x), h(x)]\) and the number \(n\), list_compose_steps should produce \([n, f(n), g(f(n)), h(g(f(n)))\]. Please note that using append
will not work in your lambda expression because `append` doesn’t return anything. To append some number `x` to a list, you should write `list+[x]` instead of `list.append(x)`.

Example

- `list_compose_steps([lambda x: x+1, lambda x: x+2, lambda x: x+3], 4)`
  → `[4, 5, 7, 10]`.

Keep in mind that you should never call `compose` and that you can only use `map` or `reduce`. Also remember that the number `n` should be the first element in your output list.

All of these should raise an `InvalidInputException` if any of the inputs are `None`.

Testing

Write your test cases in `functional_test.py`. A few examples have already been filled in for you. **DO NOT** write your tests within the example test functions we provide! Our scripts will skip the test functions we provide, so write your own functions to test your code thoroughly.