DSC 40A - Homework 1

Due: Friday, October 7, 2022 at 2:00 PM PT

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Homeworks are due to Gradescope by before the start of the lecture on the due date.

Homework will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should **always explain and justify** your conclusions, using sound reasoning. Your goal should be to convince the reader of your assertions. If a question does not require explanation, it will be explicitly stated.

Homeworks should be written up and turned in by each student individually. You may talk to other students in the class about the problems and discuss solution strategies, but you should not share any written communication and you should not check answers with classmates. You can tell someone how to do a homework problem, but you cannot show them how to do it.

For each problem you submit, you should **cite your sources** by including a list of names of other students with whom you discussed the problem. Instructors do not need to be cited.

This homework will be graded out of 60 points. The point value of each problem or sub-problem is indicated by the number of avocados shown.

Problem 1. When life gives you lemons...

Suppose you're operating a fruit stand near La Jolla Cove. Your stand only accepts cash, so you don't have access to digital receipts of all of your transactions.

a) 🥑🥑🥑🥑 You want to keep track of the mean transaction price so far during the day. Instead of writing down the price of each transaction and re-calculating the mean every time someone makes a purchase, you want to come up with a way to only keep track of the mean transaction price.

Let x_i represent the price of the ith transaction (e.g. the 5th transaction of the day is x_5). We define μ_n to be the mean of the first n transactions; that is, $\mu_n = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Determine a formula for μ_{n+1} that only uses the variables μ_n, n, and x_{n+1}. (*Hint: Start by writing out the definition of* μ_{n+1}.)

b) 🥑 Why does the above result imply that you don't need to store the values of all transactions individually?

c) 🥑🥑 So far, you've sold 9 items and the mean transaction price (including the 9th item) is \$15. How much would your **next** transaction price have to be in order for your new mean transaction price to be \$20.5?

d) 🥑🥑🥑 We've shown that it's possible to update the mean after each transaction without keeping track of all transaction prices individually. Is it possible to do the same with the mode? That is, suppose M_n represents the mode transaction price of the first n transactions. Is it possible to determine M_{n+1} using just M_n and n?

If so, give a formula for M_{n+1} in terms of just M_n and n. If not, explain why not, and provide a counterexample with two different sets of transaction prices with the same values for M_n and n but different values for M_{n+1}.

Problem 2. Garage Sale: Everything $2d$ Dollars!

Suppose you're selling n items at a garage sale. Let $x_1, x_2, \ldots, x_n$ represent the dollar values of the items you're selling. These actual dollar values are known to you. To simplify the management of your garage sale, you want to sell everything for the same fixed price of d dollars. The question is, what should this fixed price be?

Let's assume that for each item, if the asking price of the item is more than its value, then the item will not sell, and otherwise, it will sell for **two times** the asking price. Example, if you mark an an item to be \$x, it will sell for \$2x (People just tend to overpay for everything these days!)

First, we're just going to consider the simplest case where you have a single item at your garage sale, whose value is fixed at x_1 dollars, where $x_1 > 0$.

a) 🥑 Define a piece wise function $P(d)$, or $P(d; x_1)$, that represents the amount of money you'll make if you price the item at d dollars. No justification needed.

b) 🥑 Draw a graph of $P(d)$ as a function of d. Mark x_1 on the horizontal and vertical axes. No justification needed.

c) 🥑 If your goal is to maximize your income at this one-item garage sale, how should you set the asking price d? Explain how the answer based on the graph above matches up with the intuitive answer to this question.

Consider another approach, where instead of creating a function to represent the money you'd make if you priced the item at d dollars, you create a function to represent the amount of potential income you'd lose if you price the item at d dollars. For example, if an item has a value of \$5, you'd lose \$2 of potential income by pricing it at \$4.

d) 🥑 Define a piecewise function $L(d)$, or $L(d; x_1)$, that represents the potential income lost if you price the item at d dollars. No justification needed.

e) 🥑 Draw a graph of $L(d)$ as a function of d. Mark x_1 on the horizontal and vertical axes. No justification needed.

You should find that the price d that minimizes this loss function is the same price d that maximizes the money earned. This can be explained mathematically, because there is a simple relationship between $L(d)$ and $P(d)$.

f) 🥑 Find this relationship. Express $L(d)$ in terms of $P(d)$.

Informally, a minimizer of a function f is an input x^* where f achieves its minimum value. More formally, x^* is a minimizer of f if $f(x^*) \leq f(x)$ for all values of x. In the same way, x^* is a maximizer of f if $f(x^*) \geq f(x)$ for all values of x.

g) 🥑🥑🥑 Use the definitions of minimizer and maximizer given above to show that a minimizer of $L(d)$ is a maximizer of $P(d)$ **and** that a maximizer of $P(d)$ is a minimizer of $L(d)$ (make sure your solution addresses both parts). This shows that minimizing $L(d)$ is equivalent to maximizing $P(d)$.

Hint: In both directions of this proof, you should refer to your answer from part f).

Now suppose you're selling two items, whose values are x_1 and x_2, with $0 < x_1 \leq x_2$. We want to find the optimal asking prices, which is the value of d that maximizes the total money earned $P(d; x_1) + P(d; x_2)$, or equivalently, minimizes the total loss $L(d; x_1) + L(d; x_2)$.

h) 🥑🥑 Argue why the optimal asking price can never be less than x_1 or more than x_2.

i) 🥑🥑 If we want to set our asking price to be a value grater than x_1, what value should we set are asking price to, so as to make it optimal? Give a reason for your answer.

j) 🥑 Give an example of values $x_1 \neq x_2$ where the optimal asking price is $d = x_1$.

k) 🥑 Similarly, give an example of values $x_1 \neq x_2$ where the optimal asking price is $d = x_2$.

l) 🥑🥑 What condition needs to be satisfied for the optimal asking price to be $d = x_2$? In other words, can you characterize all possible solutions to the previous question?

Problem 3. Skylar's Idea

In lecture, we discussed the fact that absolute error penalizes overestimates the same amount it penalizes underestimates. (For instance, for a true salary of $100,000, predictions of $90,000 and $110,000 both have absolute errors of $10,000.)

Your friend Skylar devises a new type of error, *scaled absolute error*, which generalizes absolute error so that overestimates and underestimates can be penalized differently. Specifically, for any two scalar constants a and b, the scaled absolute error for a single data point y and prediction h defined as

$$L_{sca}(h, y) = \begin{cases} a(h - y) & y \leq h \\ b(y - h) & y > h \end{cases}$$

As usual, we will seek to minimize mean scaled absolute error, i.e.

$$R_{sca}(h) = \frac{1}{n} \sum_{i=1}^{n} L_{sca}(h, y_i)$$

First, let's suppose we have a dataset with 9 points, $\{2, 3, 7, 11, 15, 18, 19, 22, 29\}$.

a) 🥑 Find h^* for the provided dataset when $a = b = 1$.

b) 🥑🥑🥑🥑🥑 Find h^* for the provided dataset when $a = 1$ and $b = 3$. Explain how you arrived at your answer.

c) 🥑🥑 Find h^* for the provided dataset when $a = 3$ and $b = 1$.

Hint: Most of the work in this question should be in determining how to answer part b). Once you've answered that, you should be able to answer part c) much more easily.

Problem 4. Prata's Idea

In lecture, we argued that a good prediction h is one that has a small mean absolute error:

$$R(h) = \frac{1}{n} \sum_{i=1}^{n} |y_i - h|$$

We saw that the median of $y_1, \ldots, y_n$ is the prediction with the smallest mean absolute error. Your friend Prata thinks that instead of minimizing the mean absolute error, it is better to minimize the *product of the absolute errors*:

$$P(h) = \prod_{i=1}^{n} |y_i - h|$$

The above formula is written using *product notation*, which is similar to summation notation, except terms are multiplied and not added. For example,

$$\prod_{i=1}^{n} a_i = a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n$$

In this problem, we'll see if Prata has a good idea.

a) 🥑🥑 Without using a calculator or computer, graph $P(h)$ for the data set $y_1 = 4, y_2 = -1$.

b) 🥑🥑 Let's say we have three points in our data set, $y_1 = 1$, $y_2 = 5$, $y_3 = 8$. Now, lets say you built a model that makes a prediction h = 6. What would be the value of the error using Prata's error function ($P(h)$)?

c) 🥑🥑🥑🥑 For an arbitrary data set $y_1, y_2, \ldots, y_n$, what value(s) h^* minimize $P(h)$? Discuss the pros and cons of using Prata's prediction strategy. What factors about the data set or application will influence whether this prediction strategy gives good predictions?

Problem 5. More Data for Better Predictions

Note: For this problem, you will need to write code. You are free to do this locally on your computer, but you can also use `https://datahub.ucsd.edu`; there is a DSC 40A image there that you can use if you wish. You will not need to turn in any separate code files; we will specify what you need to include in your PDF submission in each subpart.

Companies seem to be hoarding as much data as they can get these days. One reason for this data hoarding is that more data enables better prediction-making. In this problem, we'll try to quantify how much better predictions can become with more data.

Let's start with a simple experiment. Imagine that our data contains randomness (as will all phenomena on earth) and are generated identically and independently from a uniform distribution within an interval on the real line: that is, for each data point, there is equal chance of it taking any value within a fixed interval.

Suppose that the width of that interval is 10, but we do not know the center of the interval. Based on the data we observe, we want to estimate, or predict, the center of the interval, which we call θ, and we want to see whether more data will help us in determining θ.

We can generate synthetic data to perform this experiment. We'll start by fixing a value θ, which will determine the interval from which we generate our data. In Python, let's first import the **numpy** package and generate a true θ parameter by picking a number randomly from 10 to 50:

```
import numpy as np
theta = np.random.uniform(10.0, 50.0)
```

Note in the above code that the function **np.random.uniform()** takes in two arguments, the lower and upper limits of an interval, and outputs a random number in that interval, chosen from a uniform distribution.

Now we have fixed the *true* model that generates data. The value of θ, or the center of the interval from which we will generate data, is determined, though we don't know what is. We've decided that our interval will have a width of 10, which means the data will be uniformly distributed across the interval $[\theta - 5, \theta + 5]$.

Next, we can generate synthetic data from our model using the **random.uniform()** function. For example, if we wish to generate 10 data points, we can set number of data $n = 10$ and write:

```
n = 10
x = np.random.uniform(theta - 5.0, theta + 5.0, n)
```

Here the additional third argument to the function **np.random.uniform()** specifies how many random numbers we want to generate. The default setting (as in our previous lines of code) is 1.

If we would like to take a look at the generated data, we can print it:

```
print(x)
```

You will see that our data **x** are stored as an array and are randomly distributed in a certain range. With the data generated, we can now start to infer what the value of θ is.

a) 🥑 Run the code given above. **By hand**, draw a number line and mark each data point in array `x`. Use your drawing to make a guess about the value of θ based on the data in `x`. Include your drawing and guess in your submission.

If we make a guess, h, how good of a guess have we made? Since we have access to the true value of θ, we can use loss functions to tell us how close our predictions are from the truth. For example, a small absolute error $|h - \theta|$ or a small squared error $(h - \theta)^2$ can mean that our guess h is close to the truth θ.

For now, we'll make a guess h and use the absolute error,

```
abs_err = abs(h-theta)
```

to assess how good of a guess we've made. Smaller values of *abs_err* correspond to better guesses.

We can also make systematic predictions for θ, instead of looking at the data and guessing. Suppose we use the mean of the data as our prediction. We can calculate the mean of the data using numpy's built-in function:

```
h = np.mean(x)
```

We can then calculate the absolute error corresponding to this prediction:

```
abs_err_mean = abs(h-theta)
```

b) 🥑🥑 Calculate the absolute loss *abs_err* corresponding to your guess in part (a). Then, calculate the absolute loss *abs_err_mean* corresponding to the mean. Did you make a better guess by looking at the data, or was the mean a better guess for θ? Include the values of your guess, *abs_err* , the mean of the `x` values, *abs_err_mean*, and the true value of `theta` in your submission.

c) 🥑🥑🥑🥑 Now try this computation for $n = 10, 20, 30, \ldots, 150$ using a for loop, using the mean as your guess each time. Observe how the squared error in the mean estimate changes as a function of the number of data values. Is it correct that more data leads to better estimates of the truth?

For this question, turn in a plot of *abs_err_mean* with n on the horizontal axis and *abs_err_mean* on the vertical axis. Use **matplotlib.pyplot** to plot the results. For example, if **abs_err_mean** were to take values of $[1, 2, 4, 8, 16, 32, 64, 128]$ when n takes values of $[1, 2, 3, 4, 5, 6, 7, 8]$, then we could make the plot using the following code:

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4,5,6,7,8], [1,2,4,8,16,32,64,128])
plt.show()
```

Hint: Our solution to this problem involves defining a function that takes in no arguments and returns an array of length 15 containing *abs_err_mean* for each value of `n`. If you write your code this way, it will make the next two parts of this problem much easier.

d) 🥑🥑🥑🥑 How does error scale with the number of data points?

To answer this question, make a similar plot as before, except for each value of n, repeat the experiment 1000 times and calculate the average *abs_err_mean*'s you obtain for each experiment. This reduces the noise and allows you to see the overall trend. Since error decreases as n increases, it may also be helpful to plot $1.0/err$.

For this question, turn in a plot of *abs_err_mean* averaged over 1000 runs and a plot of $1.0/$*abs_err_mean* averaged over 1000 runs. Then use your plots to explain how the error appears to scale as the number of data points grows. If we were to double the number of data points, say, how much would we expect the error to change?

Hint: If you followed our suggestion in the previous subpart, this subpart should not take you very long. You should call the function you wrote in the previous subpart 1000 times and average the results.

e) 🥑🥑🥑 Next, repeat the entire experiment, except use the mode of the data as your guess and use the absolute loss to asses the quality of your guess.

For this question, we would recommend using the *random.randint* function provided by **numpy** to get more overlapping values.

```
n = 10
x = np.random.randint(theta - 5.0, theta + 5.0, n)
```

For this question, turn in a plot of absolute error *abs_err_mode* averaged over 1000 runs and a plot of 1.0/*abs_err_mode* averaged over 1000 runs. Then use your plots to explain how the absolute error appears to scale as the number of data points grows. If we were to double the number of data points, say, how much would we expect the error to decrease?

Hint: look carefully at the axes of your plots.

f) 🥑🥑🥑 From the plots of the respective curves using mean of data (*abs_err_mean*) and mode of data (*abs_err_mode*) you obtained from the previous sections, which one according to you is the most useful strategy in practice and why?

You do not need to turn in any code for this problem, just a few specified numbers, your plots, and explanations.