Problem Set 8

Due: Wednesday, March 8, by 11:59pm (*except for extra credit – see below*).

Instructions

Solutions format and late policy. See PSet 1 for further details. The same requirements and policies still apply. Also follow the typesetting instructions from the prior PSets.

Collaboration policy. The written problems (Tasks 1-6) on this pset may be done with a **single partner**. In this case, only one person will submit the written part on Gradescope and add their partner as a collaborator. You must do Task 8 on your own. Task 9 is an extra credit coding (plus written) problem, which you must do on your own if you choose to do it.

Solutions submission. You must submit your solution via Gradescope. In particular:

- For the solutions to Tasks 1-6, submit under "PSet 8 [Written]" a **single** PDF file containing the solutions to Tasks 1-6 (for you and your partner). Each numbered task should be solved on its own page (or pages). Follow the prompt on Gradescope to link tasks to your pages. Do not write your names on the individual pages – Gradescope will handle that.
- Task 7 is purely optional and will not be graded, so no need to submit.
- For the programming part (Task 8), submit your code under "PSet 8 [Coding]" as a file called min.hash.py.
- Task 9 is an extra credit problem that includes coding and some written questions. If you do this part, submit it under "PSet 8 [Extra Credit – Knapsack]". Task 9 is due Saturday, March 11 by 11:59pm. (Note that Tasks 1-6 and task 8 are due Wednesday, March 8 by 11:59pm.

Task 1 – Practice with conditional expectation [10 pts]

Suppose X and Y have the following joint PMF:

X/Y	1	2	3
0	1/4	3/16	1/16
1	1/8	0	3/8

a) (3 points) what is $\mathbb{P}(X = 1 \mid Y = 2)$?

b) (3 points) What is $\mathbb{E}[X \mid Y = 2]$?

c) (4 points) What is

$$\mathbb{E}\left[\frac{X}{Y} \;\middle|\; X^2 + Y^2 \leqslant 4\right]?$$

Use the fact that if events $A_1, \ldots, A_n$ partition an event A, then

$$\mathbb{E}[F \mid A] = \sum_{i=1}^{n} \mathbb{E}[F \mid A_i]\,\mathbb{P}(A_i \mid A) = \frac{1}{\mathbb{P}(A)} \sum_{i=1}^{n} \mathbb{E}[F \mid A_i]\,\mathbb{P}(A_i)$$

for any random variable F.

Task 2 – TA positions [15 pts]

12 students have decided to apply for a TA position next quarter. The number of courses that they can be a TA for is a Poisson random variable with mean 5. Suppose that each student independently chooses exactly one of the courses to apply for uniformly at random (and independently of the choices of the other applicants). Suppose also that each student has probability 0.2 of being acceptable as a TA to the professor teaching any course they apply for. Use the law of total expectation to compute the expected number of courses that have at least one applicant acceptable to the professor for that course.

Task 3 – Lazy Grader [12 pts]

Prof. Lazy decides to assign final grades in CSE 312 by ignoring all the work the students have done and instead using the following probabilistic method: each student independently will be assigned an A with probability θ, a B with probability 3θ, a C with probability $\frac{2}{3}$, and an F with probability $\frac{1}{3} - 4\theta$. When the quarter is over, you discover that only 10 students got an A, 35 got a B, 40 got a C, and 15 got an F.

Find the maximum likelihood estimate for the parameter θ that Prof. Lazy used. Give an exact answer as a simplified fraction. You do not need to check second order conditions.

Task 4 – Continuous MLE [24 pts]

You do not need to check second order conditions in the following.

a) Let $x_1, x_2, \ldots, x_n$ be independent samples from an exponential distribution with unknown parameter λ. What is the maximum likelihood estimator for λ?

b) Given $\theta > 0$. Suppose that $x_1, \ldots, x_n$ are i.i.d. realizations (aka samples) from the model

$$f(x;\theta) = \begin{cases} \theta x^{\theta-1} & 0 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

Find the maximum likelihood estimate for θ.

Task 5 – Elections [14 pts]

Individuals in a certain country are voting in an election between 3 candidates: A, B and C. Suppose that each person makes their choice independent of others and votes for candidate A with probability θ_1, for candidate B with probability θ_2 and for candidate C with probability $1 - \theta_1 - \theta_2$. (Thus, $0 \leqslant \theta_1 + \theta_2 \leqslant 1$.) The parameters θ_1, θ_2 are unknown.
Let n_A, n_B, and n_C be the number of votes for candidate A, B, and C, respectively. What are the maximum likelihood estimates for θ_1 and θ_2 in terms of n_A, n_B, and n_C?
(You don't need to check second order conditions.)

Task 6 – (Un)biased Estimation [10 pts]

Let $x_1, \ldots, x_n$ be independent samples from Unif$(0, \theta)$, the continuous uniform distribution on $[0, \theta]$. Then, consider the estimator $\hat{\theta}_{\text{first}} = 2x_1$, i.e., our estimator ignores the samples $x_2, \ldots, x_n$ and just outputs twice the value of the first sample.
Is $\hat{\theta}_{\text{first}}$ unbiased?

Task 7 – Covariance – extra problem for your benefit only [0 pts]

We have unfortunately run out of time to cover the important concept of *covariance*. If you'd like to get ahead of the game, we highly recommend doing this problem. But it will **NOT be graded.** Solutions will be posted though.

Note that some portions of this problem are covered in Section 5.3 of the book. For any two random variables X, Y the *covariance* is defined as

$$\text{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

In this problem, if you prefer, you may assume that X and Y are discrete random variables.

a) (3 points) Show that
$$\text{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y].$$

b) (3 points) Show that for any two random variables
$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y).$$

c) (3 points) If $\mathbb{E}[Y|X = x] = x$ for all x, show that $\text{Cov}(X, Y) = \text{Var}(X)$.

d) (3 points) If X, Y are independent, show that $\text{Cov}(X, Y) = 0$.

e) (3 points) If X and Y have $\text{Cov}(X, Y) > 0$, we say that X and Y are positively correlated. If $\text{Cov}(X, Y) < 0$, we say that X and Y are negatively correlated. Suppose that $\Omega_X = \{0, 1\}$, $\Omega_Y = \{0, 1\}$ and $\Omega_{X,Y} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. Give a valid joint probability mass function for X and Y for which X and Y are positively correlated. Then give a different joint probability mass function for X and Y (same ranges) for which X and Y are negatively correlated.

Task 8 – Distinct Elements [20 pts]

Recall the setup for the $\mathrm{MinHash}$ algorithm presented in class. The universe of is the set $\mathcal{U}$ (think of this as the set of all 8-byte integers), and we have a single **uniform** hash function $h : \mathcal{U} \to [0, 1]$. That is, for an integer y, pretend $h(y)$ is a **continuous** $\mathrm{Unif}(0, 1)$ random variable. That is, $h(x_1), h(x_2), ..., h(x_N)$ for any N **distinct** elements are iid continuous $\mathrm{Unif}(0, 1)$ random variables, but since the hash function always gives the same output for some given input, if, for example, the i-th user ID, x_i, and the j-th user ID, x_j, are the same, then $h(x_i) = h(x_j)$ (i.e., they are the "same" $\mathrm{Unif}(0, 1)$ random variable).

Then, the $\mathrm{MinHash}$ algorithm is realized by the following pseudocode, which explains its two key functions:

1. $\mathrm{UPDATE}(x)$: How to update your variable when you see a new stream element.
2. $\mathrm{ESTIMATE}()$: At any given time, how to estimate the number of distinct elements you've seen so far.

Note that this differs from the syntax used on the slides, but captures the same algorithm.

MinHash Operations

```
function INITIALIZE()
    val ← ∞
function UPDATE(x)
    val ← min {val, h(x)}
function ESTIMATE() return round (1/val − 1)
for i = 1, . . . , N: do                    ▷ Loop through all stream elements
    UPDATE(x_i)                             ▷ Update our single float variable
return ESTIMATE()                           ▷ An estimate for n, the number of distinct elements.
```

To help you out with the following questions, we have set up an edstem lesson. However, you are required to upload your final solution to Gradescope (see instructions above).

a) Implement the functions UPDATE and $\mathrm{ESTIMATE}$ in the $\mathrm{MinHash}$ class of $\mathrm{min\_hash.py}$.

b) The estimator we used in **a)** has high variance, and therefore it may not always give good answer. As outlined in class, we improve this by considering k variables

$$\mathsf{val}_1, \mathsf{val}_2, \ldots, \mathsf{val}_k$$

where each of val_i, $1 \leqslant i \leqslant k$ is an i.i.d. random variable with the distribution of the minimum of $m \leqslant N$ independent $\mathrm{Unif}(0, 1)$ variables, obtained by hashing the N elements in the stream with independent hash functions $h^1, \ldots, h^k$. Our final estimate will then be

$$\hat{n} = \frac{1}{\widehat{\mathsf{val}}} - 1 \quad \text{where} \quad \widehat{\mathsf{val}} = \frac{1}{k} \sum_{i=1}^{k} \mathsf{val}_i.$$

Implement the functions UPDATE and $\mathrm{ESTIMATE}$ in the $\mathrm{MultMinHash}$ class of $\mathrm{min\_hash.py}$ using the improved estimator.

Refer to Section 9.5 of the book for more details on the distinct elements algorithm.

Task 9 – Extra Credit: Knapsacks (Coding) [28 pts]

This problem is due Saturday, March 11 by 11:59pm.

Markov Chain Monte Carlo (MCMC) is a technique that can be used to heuristically and approximately solve otherwise hard optimization problems (among other things). We will be talking about Markov chains on Friday, March 3 (and possibly also on Monday, March 6). If you want to do this extra credit problem, you will need to read Section 9.6 of the book.
Having said that, the general strategy of the MCMC technique is as follows:

1. Define a Markov Chain with states being possible solutions, and (implicitly defined) transition probabilities that result in the stationary distribution π having higher probabilities on "good" solutions to our problem. We don't actually compute π, but we just want to define the Markov Chain such that the stationary distribution would have higher probabilities on more desirable solutions.

2. Run MCMC, i.e., simulate the Markov Chain for many iterations until we reach a "good" state/solution.

In this question, there is a collection of n items, numbered 0 to $n-1$, available to us, and each has some *value* and some *weight*, both of which are positive real values. We want to find the optimal subset of items that maximizes the total value (the sum of the values of the items we take), subject to the total weight (the sum of the weights of the items we take) being less than some $W > 0$. (This is known as the **knapsack problem**). In items.txt, you'll find a list of potential items with each row containing the name of the item (string), and its value and weight (positive floats).
You will implement an MCMC algorithm which also depends on a parameter T that is not part of the problem definition. Pseudocode is provided below, and a detailed explanation is provided immediately after.

Algorithm 1 MCMC for 0-1 Knapsack Problem

subset ← vector of n zeros (indexed by 0 to $n-1$), where subset is always a binary vector in $\{0,1\}^n$ that represents whether or not we have each item. (This means that we initially start with an empty knapsack).
best_subset ← subset
for $t = 1, \ldots,$ NUM_ITER **do**
    k ← a uniformly random integer in $\{0, 1, \ldots, n-1\}$.
    new_subset ← subset but with subset$[k]$ flipped ($0 \to 1$ or $1 \to 0$).
    Δ ← value(new_subset) − value(subset)
    if new_subset satisfies weight constraint (total weight $\leqslant W$) **then**
        if $\Delta > 0$ OR ($T > 0$ AND Unif$(0,1) < e^{\Delta/T}$) **then**
            subset ← new_subset
    if value(subset) > value(best_subset) **then**
        best_subset ← subset

The extra parameter T in the MCMC algorithm represents a "temperature" that controls the trade-off between exploration and exploitation. The state space $\mathcal{S}$ is the set of all subsets of n items. The algorithm starts with a state (current subset) corresponding to an empty knapsack. At each iteration, the algorithm proposes a new state (proposed subset) as follows: choose a random index k from $\{0, 1, \ldots, n-1\}$. If the item k is not already in the knapsack (current subset), then this proposed subset will just add item k, but if item k is already in the knapsack (current subset), the proposed subset will just remove item k from the knapsack (current subset).

- If the proposed subset is infeasible (doesn't fit in our knapsack because of the weight constraint), we return to the start of the loop and abandon the newly proposed subset.

- Suppose that the proposed subset is feasible. If the proposed subset has higher total value (is better) than the current subset, we will always transition to it (exploitation). Otherwise, if it is worse and $T > 0$, with probability $e^{\Delta/T}$, we update the current subset to the proposed subset, where $\Delta < 0$ is the decrease in total value. This allows us to transition to a "worse" subset occasionally (exploration), and get out of local optima! Repeat this for NUM_ITER transitions from the initial state (subset), and output the highest value subset found during the entire process (which may not be the final subset).

a) What is the size of the Markov Chain's state space $\mathcal{S}$ (the number of possible subsets)?

b) Let's try to figure out what the temperature parameter T does.

A. Suppose that $T = 0$. Will we ever get to a worse subset than before as we transition?

B. Suppose that $T > 0$.

i. For a fixed T, does the probability of transitioning to a worse subset increase or decrease with larger absolute values of Δ (larger absolute values means "more negative" values, since $\Delta < 0$)?

ii. For a fixed Δ, does the probability of transitioning to a worse subset increase or decrease with larger values of T?

iii. Explain briefly how the temperature parameter T controls the degree of exploration we do.

c) Implement the functions value, weight, and mcmc in mcmc_knapsack.py. To this end, you will use the edstem lesson. Remember that only code submitted via Gradescope will be graded.

Hints: To get full score, you **must** use np.random.rand() to generate an uniform value in $[0, 1]$, and np.random.randint(low (inclusive), high (exclusive)) to generate your random index(es). Make sure to read the documentation and hints provided!

One item that is not required or graded that we've included in the lesson for your interest is that we have called the make_plot function to make a plot where the x-axis is the iteration number, and the y-axis is the current knapsack value (not necessarily the current best), for ntrials=10 different runs of MCMC. The plots yield interesting phenomena - for example one can imagine tuning things to choose T that will most reliably produce high knapsack values.