COMS 4232: Advanced Algorithms | Sep 6, 2023

Self-Evaluation Test: Solutions

Instructor: *Alex Andoni*

This is a self-evaluation test for you to confirm that you have the sufficient background for the class, and identify potential parts to brush up before the class. You are expected to understand and solve all problems on this test. You do not have to turn in solutions.

The main prerequisite for the class is mathematical maturity, i.e., being able to follow and write rigorous mathematical proofs, of both combinatorial and analytical flavors. In terms of concrete topics, you should be comfortable with linear algebra, minimal probability theory, some basic algorithmic notions. As a result, the problems below are partitioned by category.

1 Math

1.1 Norms

For a vector $x \in \mathbb{R}^n$, and an integer $p \in (0, \infty]$, the p-norm of x is defined as $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$.[1] Prove the following:

- $\|x\|_2 \le \|x\|_1 \le \sqrt{n}\|x\|_2$, and this is tight (i.e., each inequality becomes an equality for some non-zero x);
  - **For $\|x\|_2 \le \|x\|_1$:**
    Let e^i denote the vector of 0s with a unique 1 in the ith coordinate. So, for $x = (x_1, \dots, x_n)$ we have $x = \sum_{i=1}^n x_i \cdot e^i$. Moreover for each product in the sum, $\|x_i \cdot e^i\|_2 = \sqrt{\sum_{j=1}^n |x_i \cdot e^i_j|^2} = \sqrt{0 + \cdots + 0 + |x_i|^2 + 0 + \cdots + 0} = |x_i|$.
    Then, by the triangle inequality:
    $$\begin{aligned}\|x\|_2 &\le \sum_{i=1}^n \|x_i \cdot e^i\|_2 \\ &= \sum_{i=1}^n |x_i| \\ &= \|x\|_1\end{aligned}$$
    To see that this is tight for any dimension n, consider $x = (a, 0, \dots, 0) \in \mathbb{R}^n$ for any $a > 0$. Then, $\|x\|_2 = \sqrt{|a|^2 + 0 + \cdots + 0} = a$ and $\|x\|_1 = |a| + 0 + \cdots + 0 = a$.
  - **For $\|x\|_1 \le \sqrt{n}\|x\|_2$:**

[1] For $p = \infty$, $\|x\|_\infty$ is formally defined to be $\max_{i \in [n]} |x_i|$, which can be seen as the limit as $p \to \infty$.

By Cauchy-Schwarz (applied for second line below):

$$\begin{aligned}\|x\|_1^2 &= \left(\sum_{i=1}^n |x_i|\right)^2 \\ &= \left(\sum_{i=1}^n 1 \cdot |x_i|\right)^2 \\ &\le \left(\sum_{i=1}^n 1^2\right)\left(\sum_{i=1}^n |x_i|^2\right) \\ &= n\|x\|_2^2\end{aligned}$$

Therefore, $\|x\|_1 = \sqrt{\|x\|_1^2} \le \sqrt{n\|x\|_2^2} = \sqrt{n}\|x\|_2$. To see that this is tight for any dimension n, consider $x = (a, \dots, a) \in \mathbb{R}^n$ for any $a > 0$. Then, $\|x\|_1 = n \cdot a$ and $\|x\|_2 = \sqrt{n \cdot a^2} = \sqrt{n} \cdot a$.

- when $p = \log_2 n$, $\|x\|_p$ is a 2 approximation to $\|x\|_\infty$ (i.e., $\|x\|_\infty \le \|x\|_p \le 2\|x\|_\infty$).
  - **For** $\|x\|_\infty \le \|x\|_p$**:**
    By definition $\|x\|_\infty = \max_i |x_i|$. Note that $\max_i |x_i|^p \le \sum_{i=1}^n |x_i|^p$ because all summands are non-negative. Therefore, $(\max_i |x_i|^p)^{1/p} \le \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$ because the pth root function is monotonically increasing on non-negative numbers. Putting it all together we get,

    $$\|x\|_\infty = \left(\max_i |x_i|^p\right)^{1/p} \le \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} = \|x\|_p.$$

    Again, to see this is tight consider $x = (a, 0, \dots, 0) \in R^n$ for any $a > 0$.
  - **For** $\|x\|_\infty \le 2\|x\|_\infty$**:**
    Notice that $\sum_{i=1}^n |x_i|^p \le \sum_{i=1}^p (\max_j |x_j|)^p = n\|x\|_\infty^p$. Also, by the definition of p we have $n^{1/p} = \left(2^{\log_2 n}\right)^{1/p} = (2^p)^{1/p} = 2$.
    So,

    $$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} \le (n\|x\|_\infty^p)^{1/p} = n^{1/p}\|x\|_\infty = 2\|x\|_\infty$$

In the spirit of above, how can you relate the 10-norm of x to the 5-norm of x?
$\|x\|_{10} \le \|x\|_5 \le n^{1/10}\|x\|_{10}$. In general, to prove relations between p-norms, use Holder's inequality, a generalization of Cauchy Schwarz. In this particular case we can just apply the ℓ_1 and ℓ_2 relation for the vector y where $y_i = |x_i|^5$.

1.2 Runtime of Merge-Sort

Merge-Sort is an algorithm for sorting n numbers, which proceeds as follows. Suppose the input is an array $A[1 \dots n]$ of length n (which, for simplicity, assume is a power of 2). We partition the array into two, $A[1 \dots n/2]$ and $A[n/2 + 1 \dots n]$. We sort each part recursively and then merge the 2 (sorted) parts.

It is known that the latter merge operation can be accomplished in time $O(n)$.[2] The base case of the Merge-Sort is when $n = 1$, in which case we do nothing (the array is already sorted).

Here, you are to determine whether the following statement/proof is correct or not (why or why not).

Claim 1. *The runtime of the above MergeSort algorithm is $O(n)$.*

Proof. Let $T(n)$ be the runtime of Merge-Sort on an array of length n. Then, by the above description we have that:

$$T(n) = 2 \cdot T(n/2) + O(n),$$

corresponding to the two recursive calls and the merge operation respectively.

We prove the claim that $T(n) = O(n)$ by induction. Indeed, we have that:

$$T(n) = 2 \cdot T(n/2) + O(n) = 2 \cdot O(n/2) + O(n) = O(n).$$

This completes the proof of the claim. $\square$

The claim and the proof are wrong (in fact, $T(n) = \Theta(n \cdot \log n)$ for Merge-Sort). The error is due to a sloppy use of the $O(\cdot)$ notation.

Using the definition of $O(n)$, the claim ask to prove that: there exists n_0, c such that $T(n) \le cn$ for $n \ge n_0$. So suppose this were true inductively for $T(n/2)$. In particular, whenever $n \ge 2n_0$, we have $T(n/2) \le cn/2$. Now using the recurrence we have $T(n) = 2T(n/2) + O(n) = 2cn/2 + O(n)$, where $O(n)$ is the runtime of the merge operation, which consider to be, say, $\le 10n$. While the above sum is indeed $\le cn + 10n = (c + 10)n = O(n)$, this is not enough for the inductive hypothesis for n! In particular, the inductive hypothesis is that $T(n) \le cn$. In particular the implicit constant of the $O(\cdot)$ notation grows with the number of iterative steps.

2 Linear algebra

2.1 Quadratic forms

Let A be a n-by-n symmetric matrix whose maximum eigenvalue is 42. What is the minimum and maximum possible value for the quadratic form $x^T Ax$, where $x \in \mathbb{R}^n$ are unit-norm vectors?

Recall that this means that there exists a matrix U and diagonal matrix Σ (with the eignvalues, $\sigma_1, \sigma_2, \ldots$, on the diagonal in descending order) such that $A = U\Sigma U^T$. Note that because U is unitary, for any unit-norm x, $u^T = x^T U$ is a unit-norm vectors (and furthermore any u^T is realizable). Therefore, the maximum and minimum possible values are $\max_{u:\|u\|_2=1} \sum_{i=1}^n \sigma_i u_i u_i = \max_i \sigma_i = 42$ and $\min_{u,v:\|u\|_2=1} \sum_{i=1}^n \sigma_i u_i u_i = \min_i \sigma_i$. For arbitrary matrix A, the argmin can be as low as desired.

For the second part, we note that $A^2 = U\Sigma^2 U^T$ and hence argmax is 42^2 and argmin is $\min_i \sigma_i^2 \ge 0$. Hence the lowest it can ever be is 0.

2.2 Linear systems

Suppose A is a n-by-n real matrix and $b \in \mathbb{R}^n$. For the unknown vector $x \in \mathbb{R}^n$, describe what are the possible sets of solutions for the following:

[2] Remember that $O(f(n))$ is the set of functions $g : \mathbb{N} \to \mathbb{N}$ such that there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that, for $n \ge n_0$, $g(n) \le c \cdot f(n)$.

- $Ax = b$;

  The solution set here is either an affine subspace of $\mathbb{R}^n$ or empty. If $Ax = b$ has at least one solution, $v : Av = b$, (i.e. is consistent), then this affine subspace has the same dimension of as $\ker(A)$ (if $x \in \ker(A)$, then $A(x + v) = Ax + Av = 0 + b$). Otherwise, the system is inconsistent (i.e. $\text{rank}(A) < \text{rank}([A|b])$) and it is simply the empty set.

- $\text{argmin}_{x\in\mathbb{R}^n} \|Ax - b\|_2$.

  The set $S = \{y : y = Ax - b\}$ is again an affine subspace of $\mathbb{R}^n$ (but, unlike above, it cannot be empty). But more importantly, S is convex. Additionally, $\| \cdot \|_2$ is a convex function. Therefore this is a convex optimization problem and it has a unique minimizer. Or in other words, set always consists of a single vector $y^* \in \mathbb{R}^n$, the shortest one in S. Hence the set of solutions to argmin is the set of x's satisfying $Ax - b = y^*$. This set is either a singleton or an affine subset as per above (note that, in constrast to above, there's always at least one solution x).

To give a better idea of what's asked, consider the "$Ax = b$" case when $n = 1$ (i.e., just a simple equation). Here's the answer for this case:

The possible sets of solutions are: 1) there may not be a solution (empty set), 2) may be exactly one solution, or 3) any $x \in \mathbb{R}$ is a solution.

The above 3 cases correspond to when 1) $A = 0$, $b \neq 0$, 2) $A \neq 0$, 3) $A = b = 0$.

3 Probability

3.1 Mean estimation

Let p be a probability distribution over $[0, 1]$. The goal is to estimate the mean μ of the distribution p, up to an additive error ϵ, where $\epsilon \in (0, 1)$ is given in advance. How many samples suffice to output some $\hat{\mu}$ such that $\Pr[|\hat{\mu} - \mu| \leq \epsilon] \geq 0.9$?

Assuming, of course, that samples are independent, our algorithm will simply be to take the empirical mean of sufficiently many samples. Let X_i be the random variable corresponding to the ith sample. Then clearly, $\mu = \mathbb{E}[X_i]$ for all i. Let $\hat{\mu} = \frac{\sum_{i=1}^m X_i}{m}$ ($\mu = \mathbb{E}[\hat{\mu}]$). By Chebyshev bound, we have

$$\Pr[|\mu - \hat{\mu}| \geq \epsilon] \leq \frac{\mathbb{E}[(\mu-\hat{\mu})^2]}{\epsilon^2} \leq k^{-1} \cdot \epsilon^2$$

Hence $k = 10/\epsilon^2$ samples suffice.

3.2 Hashing

A function $h : U \to [m]$, where U is a discrete set (e.g., integers of $O(\log n)$ bits), and $m \in \mathbb{N}$, is called a hash function with m buckets. (One can see this as assigning one of the m buckets to each possible item from U.) Most common hash function is a random hash function: each $h(i)$, $i \in U$, is chosen iid at random from $[m]$.

Below, consider a set $S \subset U$ of size n.

- Let h be a random hash function with $m = n$. What is the expected size of a bucket, i.e., $|h^{-1}(i) \cap S|$ for any $i \in [m]$?

  For $x \in S$, X_x denote the indicator random variable for $h(x) = i$. Because h is random, $\mathbb{E}[X_x] = \Pr_h[h(x) = i] = 1/m$ for all $x \in S$. Moreover, size of the ith bucket is simply the sum over all X_x's. Then, via linearity of expectation, the expected load of i is $\mathbb{E}[\sum_{x\in S} X_x] = \sum_{x \in S} \mathbb{E}[X_x] = n/m$.

- A collision is an event where distinct $x, y \in S$ satisfy $h(x) = h(y)$. What is the expected number of collisions among elements of S under h? (To give an example, if one has precisely three distinct items $x, y, z \in S$ falling into the same bucket, they generate 3 collisions: (x, y), (y, z), and (z, x)).

  For $x \neq y \in S$, let $Z_{x,y}$ denote the indicator random variable for the event that $h(x) = h(y)$. Then $\mathbb{E}[Z_{x,y}] = \Pr_h[h(x) = h(y)] = 1/m$ — fix any $h(x)$, then because $y \neq x$ and h is a random function there is a $1/m$ probability h maps y to $h(x)$. By linearity of expectation, we can count collisions by simply summing $Z_{x,y}$ over distinct pairs. We have that the expected number of collisions is

  $$\mathbb{E}[\sum_{x,y\in S: x\neq y} Z_{x,y}] = \sum_{x,y\in S: x\neq y} \mathbb{E}[Z_{x,y}] = \binom{n}{2}/m = \frac{n(n-1)}{2m}$$

  . When $n = m$, this is $\Theta(n)$ collisions.

A family $\mathcal{H}$ of hash functions h is k-wise independent if for any distinct $a_1, \dots a_k \in U$ and (not necessarily distinct) $q_1, \dots q_k \in [m]$, we have that

$$\Pr_{h\in\mathcal{H}}[h(a_1) = q_1 \wedge \dots \wedge h(a_k) = q_k] = 1/m^k.$$

Note that the aforementioned random hash function is $|U|$-wise independent (or, to be more precise, the family $\mathcal{H}$ of *all* functions $h : U \to [m]$ is $|U|$-wise independent).

- How do the estimates from above change if our hash function h is chosen from some 2-wise independent hash family $\mathcal{H}_2$?

  Nothing changes. Note that in both of the above we are simply using linearity of expectation to reduce computation of the big expectation to the expectation of our indicator variables. So, it is simply a matter of showing that the expectations of X_x and $Z_{x,y}$ are unchanged if h is chosen from pairwise independent family (instead of randomly). Then via inclusion-exclusion, $\mathbb{E}[X_x] = \sum_{j=1}^{m} \Pr[h(x) = i \wedge h(y) = j] = 1/m$ and $\mathbb{E}[Z_{x,y}] = \sum_{i=1}^{m} \Pr_h[h(x) = h(y) = i] = 1/m$.

4 Algorithms

4.1 Graphs

A graph G has n nodes and m edges. Suppose that $m = n/2$ (assuming n is even). What is the smallest and the largest number of connected components that G can have?

First, note that any graph with $n/2$ to edges can be constructed iteratively in $n/2$ rounds where in each round first exactly two nodes are added followed by exactly one edge.

- **The minimal graph has $n/2$ connected components:**

Suppose not, then let G be a counter example of minimal size, n, with $c^* < n/2$ connected components. Consider the iterative construction of G, in the last step nodes $\{v_1, v_2\}$ are added with edge e. By the minimality of G, the graph G' where v_1, v_2 and e have been removed from G must have $\geq (n-2)/2 = n/2 - 1$ connected components. Now consider adding v_1, v_2, e to G'. After adding v_1, v_2 there are $n/2 - 1 + 2 = n/2 + 1$ connected components as both vertices are new. However, at best, e can connect at most two components into one leaving $n/2 + 1 - 1 = n/2$ components remaining. This contradicts our assumption on G.

The bipartite graph with $V = (\{a_1, \ldots, a_{n/2}\}, B = \{a_1, \ldots, a_{n/2}\})$ and $E = \{(a_i, b_i) : i \in [n/2]\}$ demonstrates that this is tight.

- **The maximal graph has $n - \lceil \frac{1+\sqrt{1+4n}}{2} \rceil + 1$ connected components:**

  Again, we will show that the greedy construction that yields a (partial) k-clique alongside $n - k_n$ isolated nodes where k_n is the smallest number of nodes that can contain $n/2$ distinct edges, $k_n := \text{argmin}_{k \in \mathbb{N}} \binom{k}{2} \geq n/2)$ is optimal. Solving for k_n via the quadratic formula we get $k_n = \lceil \frac{1+\sqrt{1+4n}}{2} \rceil$.

  By the above, when considering an iterative construction of a graph G, after each round there are 2 options: (a) the number of connected components increased by 1 (the edge joined two connected components, after adding two the via isolated nodes), (b) the number of connected components increased by 2 (the edge was within a pre-existing connected component).

  Suppose not, then let G be a minimal counter-example (on n nodes) with $c^* > n - k_n + 1$ connected components (for k_n defined as above). Note that k_n is monotonically increasing (with slope at most 1/2 for $n \geq 2$). At most 2 components were added in the last round of constructing G, so the penultimate G' (on $n-2$ nodes) has at least $c^* - 2 > (n - k_n + 1) - 2 \geq (n-2) - k_{n-2} + 1$ components, which contradicts the minimality of G.

4.2 Alice tells Bob

Consider Alice holds a number n and wants to communicate it to Bob *approximately* using minimal possible communication. In particular, Bob should be able to output some n' satisfying $n \leq n' \leq 2n$. How many bits does Alice need to send to Bob to achieve this goal? (Answer up to $O(\cdot)$ is enough.)

The answer is $O(\log \log n)$ bits suffice as shown by the following protocol.

- **Alice:** On input n, send the index of the most significant bit of n, $\text{msb}(n)$ (i.e. the unique number m such that $2^m \leq n < 2^{m+1}$). (Note that the length of the binary representation of n is at most $\log n$. Therefore, the largest index of this representation is can be represented with $\log \log n$ bits.)

- **Bob:** Upon receiving a binary representation of a number m, output 2^m.

Does the answer change if it's a "dialogue" — Alice and Bob sequentially communicate to each other a number of bits (think of it as if Bob can "ask questions") — and the "total communication" is the total number of bits exchanged ?

Interaction does not help in this case and the complexity remains the same because Alice can simply simulate a virtual Bob in her head (because Bob has no input, so Alice knows everything about him). In particular, suppose there was a cheaper (deterministic) interactive protocol. Then, we can compile this into a non-interactive protocol by having Alice simulate the entire interaction and send the transcript to Bob. Bob then plays the transcript for himself and outputs in accordance with the interactive protocol.