New York University Tandon School of Engineering
Computer Science and Engineering

# CS-GY 6763: Homework 2.
Due Tuesday, October 17th, 2023, 11:59pm ET.

*Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list collaborators for each problem separately, or write "No Collaborators" if you worked alone.*

## Problem 1: Hashing around the clock.

**(15 pts)** In modern systems, hashing is often used to distribute data items or computational tasks to a collection of servers. What happens when a server is added or removed from a system? Most hash functions, including those discussed in class, are tailored to the number of servers, $n$, and would change completely if $n$ changes. This would require rehashing and moving all of our $m$ data items, an expensive operation.

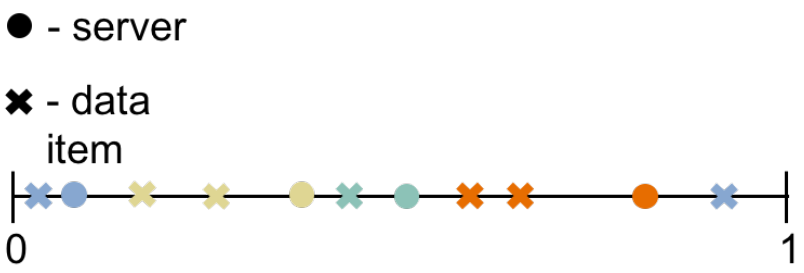



Figure 1: Each data item is stored on the server with matching color.

Here we consider an approach to avoid this problem. Assume we have access to a completely random hash function that maps any value $x$ to a real value $h(x) \in [0, 1]$. Use the hash function to map *both* data items and servers randomly to $[0, 1]$. Each data item is stored on the first server to its right on the number line (with wrap around – i.e. a job hashed below 1 but above all serves is assigned to the first server after 0). When a new server is added to the system, we hash it to $[0, 1]$ and move data items accordingly.

1. Suppose we have $n$ servers initially. When a new server is added to the system, what is the expected number of data items that need to be relocated?

2. Show that, with probability $> 9/10$, no server "owns" more than an $O(\log n/n)$ fraction of the interval $[0, 1]$. **Hint:** This can be proven without a concentration bound.

3. Show that if we have $n$ servers and $m$ items and $m > n$, the maximum load on any server is no more than $O(\frac{m}{n} \log n)$ with probability $> 9/10$.

## Problem 2(a): Analyzing Sign-JL and JL for Inner Products

**(15 pts)** Often practitioners prefer JL matrices with discrete random entries instead of Gaussians because they take less space to store and are easier to generate. We analyze one construction below.

Suppose that $\mathbf{\Pi}$ is a "sign Johnson-Lindenstrauss matrix" with $n$ columns, $k$ rows, and i.i.d. $\pm 1$ entries scaled by $1/\sqrt{k}$. In other words, each entry in the matrix has values $-1/\sqrt{k}$ with probability $1/2$ and $1/\sqrt{k}$ with probability $1/2$.

1. Prove that for any vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbb{E}[\|\mathbf{\Pi x}\|_2^2] = \|\mathbf{x}\|_2^2$ and that $\text{Var}[\|\mathbf{\Pi x}\|_2^2] \leq \frac{2}{k}\|\mathbf{x}\|_2^4$. This is the meat of the problem and will take some effort.

2. Use the above to prove that $\Pr\left[\left|\|\mathbf{\Pi x}\|_2^2 - \|\mathbf{x}\|_2^2\right| \geq \epsilon\|\mathbf{x}\|_2^2\right] \leq \delta$ as long as we choose $k = O\left(\frac{1/\delta}{\epsilon^2}\right)$. Note that this bound almost matches the distributed JL lemma proven in class, but with a worse failure probability dependence of $1/\delta$ in place of $\log(1/\delta)$.

*With more work, it's possible to improve the dependence to* $\log(1/\delta)$ *for the sign-JL matrix, but we won't do so here.*

3. Generalize your analysis above to show that JL matrices are also useful in approximating inner products between two vectors. For vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ prove that $\Pr\left[|\langle \mathbf{\Pi x}, \mathbf{\Pi y}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle| \geq \epsilon \|\mathbf{x}\|_2 \|\mathbf{y}\|_2\right] \leq \delta$ as long as we choose $k = O\left(\frac{1/\delta}{\epsilon^2}\right)$.

   *This result can also be improved to have a* $\log(1/\delta)$ *dependence in place of* $1/\delta$. .

## Problem 2(b): Join Size Estimation

**(5 pts)** One powerful application of sketching is in database applications. For example, a common goal is to estimate the *inner join size* of two tables without performing an actual inner join (which is expensive, as it requires enumerating the keys of the tables). Formally, consider two sets of unique keys $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ which are subsets of $1, 2, \ldots, U$. Our goal is to estimate $|X \cap Y|$ based on small space compressions of $X$ and $Y$.

Using your result from Problem 1, describe a method based on inner product estimation that constructs independent sketches of $X$ and $Y$ of size $k = O\left(\frac{1}{\epsilon^2}\right)$ and from these sketches can return an estimate $Z$ for $|X \cap Y|$ satisfying

$$|Z - |X \cap Y|| \leq \epsilon \sqrt{|X||Y|}$$

with probability 9/10.

## Problem 3: Compressed classification.

**(10 pts)** In machine learning, the goal of many classification methods (like support vector machines) is to separate data into classes using a *separating hyperplane*.

Recall that a hyperplane in $\mathbb{R}^d$ is defined by a unit vector $a \in \mathbb{R}^d$ ($\|a\|_2 = 1$) and scalar $c \in \mathbb{R}$. It contains all $h \in \mathbb{R}^d$ such that $\langle a, h\rangle = c$.

Suppose our dataset consists of $n$ unit vectors in $\mathbb{R}^d$ (i.e., each data point is normalized to have norm 1). These points can be separated into two sets $X, Y$, with the guarantee that there exists a hyperplane such that every point in $X$ is on one side and every point in $Y$ is on the other. In other words, for all $x \in X, \langle a, x\rangle > c$ and for all $y \in Y, \langle a, y\rangle < c$.

Furthermore, suppose that the $\ell_2$ distance of each point in $X$ and $Y$ to this separating hyperplane is at least $\epsilon$. When this is the case, the hyperplane is said to have "margin" $\epsilon$.

1. Show that this margin assumption equivalently implies that for all $x \in X, \langle a, x\rangle \geq c + \epsilon$ and for all $y \in Y, \langle a, y\rangle \leq c - \epsilon$.

2. Show that if we use a Johnson-Lindenstrauss map $\Pi$ to reduce our data points to $O(\log n/\epsilon^2)$ dimensions, then the dimension reduced data can still be separated by a hyperplane with margin $\epsilon/4$, with high probability (say $> 9/10$).

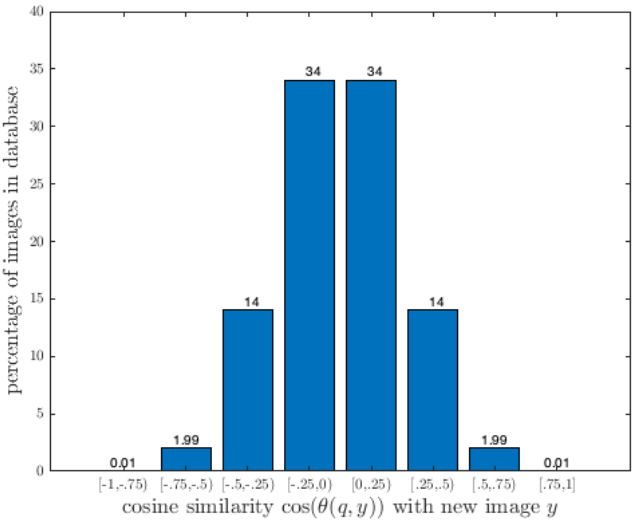
## Problem 4: LSH in the Wild

*This exercise does not involve formal proofs or analysis like more typical problem set problems. It will likely involve some coding or spreadsheet work.*

**(10 pts)** To support its largely visual platform, Pinterest runs a massive image de-duplication operation built on Locality Sensitive Hashing for Cosine Similarity. You can read about the actual system here. All information and numbers below are otherwise purely hypothetical.

Pinterest has a database of $N =$ **1 billion** images. Each image in the database is pre-processed and represented as a vector $\mathbf{q} \in \mathbb{R}^d$. When a new image is pinned, it is also processed to form a vector $\mathbf{y} \in \mathbb{R}^d$. The goal is to check for any existing duplicates or near-duplicates to $\mathbf{y}$ in the database. Specifically, Pinterest would like to flag an image $\mathbf{q}$ as a near-duplicate to $\mathbf{y}$ if $\cos(\theta(\mathbf{q}, \mathbf{y})) \geq .98$. We want to find any near-duplicate with probability $\geq 99\%$.

Given this requirement, your job is to design a multi-table LSH scheme using SimHash to find candidate near-duplicates, which can then be checked directly against $\mathbf{y}$. To support this task, Pinterest has collected data on the empirical distribution of $\cos(\theta(\mathbf{q}, \mathbf{y}))$ for a typical new image $\mathbf{y}$. It roughly follows a bell-curve:




Pinterest wants to consider two possible computational targets for your LSH scheme, which will determine the speed of the de-duplication algorithm:

1. Ensure that no more than 1 million candidate near-duplicates are checked on average when a new image is pinned. "Checked" means that the image's cosine similarity with the new image is computed explicitly, which is a computationally expensive operation.
2. Ensure that no more than $200,000$ candidates are checked on average when a new image is pinned.

Based on the data above, describe how to set parameters for your LSH scheme to minimize the space (i.e., number of tables) used, while achieving each of the above goals. Justify your answers, and any assumptions you make. If you code anything up to help calculate your answer, please attach the code. As in lecture, you can assume that each hash table has $m = O(N)$ slots and this is large enough to ignore lower order terms depending on $1/m$.

## Extra Credit: Revisiting Wikipedia Size Estimation

**(10 pts)** Many students observed that, when trying to estimate the number of articles on Wikipedia in the last homework, the mark-and-recapture method consistently returned an estimate below the claimed number of articles on Wikipedia. In this problem we will try to understand why. My guess is that the underestimate is due to the fact that Wikipedia's random article generator does not return *truly uniform* random articles. As discussed here, Wikipedia assigns each article $i$ a random id $r_i$, which we can model as a random real number in $[0, 1]$. Then, to pick a random article, a random real number in $[0, 1]$ is sampled and article $i$ is return if that number lies in the range $[r_i, r_{i+1}]$. That is, the range $[0, 1]$ is partitioned into $n$ intervals and the probability of returning article $i$ is equal to the length of the $i^{\text{th}}$ interval, which we denote by $p_i$. Since these intervals themselves are random, the probability distribution won't be *perfectly* uniform.

1. Prove that, when the interval lengths are not perfectly uniform, the mark-and-recapture method should be expected to underestimate the number of articles. **Hint:** Prove that the expected number of collisions $D$ after $m$ samples equals $\binom{m}{2} \sum_{i=1}^{n} p_i^2$, and show that this expectation is minimized when $p_1 = \ldots = p_n = \frac{1}{n}$. So, we get too many collisions for a non-uniform distribution, causing an underestimate of $n$.
2. Show that, if Wikipedia uses the scheme above, we expect that the mark-and-recapture method will systematically underestimate the number of articles by almost exactly a factor of two. **Hint:** Try to prove a bound on $\mathbb{E}\left[\sum_{i=1}^{n} p_i^2\right]$.