8.1.2025 ***Adaptive data analysis***

Lecture 9: The L_2 Heavy Hitters Problem

Source: Lecture notes by Aaron Roth and Adam Smith

Lecturer: Uri Stemmer

Recall the formulation of the heavy-hitters streaming problem we studied in Lecture 7:

Given a (prefix of a) stream $S = (s_1, s_2, \dots, s_i)$, where each $s_\ell \in [n]$, return a list of size k containing all elements that appear more than i/k times in the stream, i.e., all elements $x \in [n]$ such that

$$\text{weight}_i(x) \triangleq |\{\ell \in [i] : s_\ell = x\}| > \frac{i}{k}$$

We saw a deterministic algorithm with space $\approx k$ for this problem, which is optimal in the sense that every algorithm for this problem must use $\Omega(k)$ space in the worst-case (as there could be k "heavy" elements that the algorithm "must remember").

However, there is still room for improvement, in the sense that if no element has weight i/k then the algorithm from Lecture 7 guarantees nothing. That is, even if we are OK with paying space $\approx k$, we could still hope to "catch" elements with smaller weights under some conditions.

Let's begin with the following more general formulation of the problem that allows for *deletions* in the stream.

Definition [The L1 Heavy-Hitters Problem]:

- Every update in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- After $\boldsymbol{t}$ updates, denote the frequency vector of the stream as
$$\boldsymbol{f^{(t)} = \Delta_1 \cdot u_1 + \cdots + \Delta_t \cdot u_t}$$
- At query time, say after $\boldsymbol{t}$ updates, the goal is to return a set of $\boldsymbol{O(k)}$ coordinates (called keys) that include all the L1-heavy hitters of $\boldsymbol{f^{(t)}}$, that is, all keys $\boldsymbol{i} \in [\boldsymbol{n}]$ such that
$$\boldsymbol{\left| f^{(t)}[i] \right| > \frac{1}{k}\left\| f^{(t)} \right\|_1 = \frac{1}{k}\left(\left| f^{(t)}[1] \right| + \cdots + \left| f^{(t)}[n] \right| \right)}$$
- The output is correct when all heavy hitters are reported

The L1 heavy hitters problem can be solved deterministically with small space (even though in Lecture 7 we did not address deletions). Now consider the following variant, where we aim to identify elements which are "heavy" w.r.t. the L2 norm of the frequency vector rather than w.r.t. the L1 norm:

Definition [The L2 Heavy-Hitters Problem]:

- Every update in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- After $\boldsymbol{t}$ updates, denote the frequency vector of the stream as
$$\boldsymbol{f^{(t)} = \Delta_1 \cdot u_1 + \cdots + \Delta_t \cdot u_t}$$
- At query time, say after $\boldsymbol{t}$ updates, the goal is to return a set of $\boldsymbol{O(k)}$ coordinates (called keys) that include all the L2-heavy hitters of $\boldsymbol{f^{(t)}}$, that is, all keys $\boldsymbol{i} \in [\boldsymbol{n}]$ such that
$$\boldsymbol{\left| f^{(t)}[i] \right|^2 > \frac{1}{k}\left\| f^{(t)} \right\|_2^2 = \frac{1}{k}\left(\left| f^{(t)}[1] \right|^2 + \cdots + \left| f^{(t)}[n] \right|^2 \right)}$$
- The output is correct when all heavy hitters are reported

Note:

(1) There could be at most $\boldsymbol{k}$ L2-heavy keys, as otherwise $\left\|\boldsymbol{f}^{(t)}\right\|_2^2 \geq \left(\left|\boldsymbol{f}^{(t)}[\boldsymbol{i_1}]\right|^2 + \cdots + \left|\boldsymbol{f}^{(t)}[\boldsymbol{i_{k+1}}]\right|\right) \geq \left(\frac{1}{\boldsymbol{k}}\left\|\boldsymbol{f}^{(t)}\right\|_2^2 + \cdots + \frac{1}{\boldsymbol{k}}\left\|\boldsymbol{f}^{(t)}\right\|_2^2\right) = \frac{\boldsymbol{k+1}}{\boldsymbol{k}}\left\|\boldsymbol{f}^{(t)}\right\|_2^2$

(2) If a key $\boldsymbol{i}$ is L1-heavy with parameter $\sqrt{\boldsymbol{k}}$ then it must also be L2-heavy with parameter $\boldsymbol{k}$ because

$$\left|\boldsymbol{f}^{(t)}[\boldsymbol{i}]\right| > \frac{\mathbf{1}}{\sqrt{\boldsymbol{k}}}\left\|\boldsymbol{f}^{(t)}\right\|_1 \quad \Rightarrow \quad \left|\boldsymbol{f}^{(t)}[\boldsymbol{i}]\right|^2 > \frac{\mathbf{1}}{\boldsymbol{k}}\left\|\boldsymbol{f}^{(t)}\right\|_1^2 \geq \frac{\mathbf{1}}{\boldsymbol{k}}\left\|\boldsymbol{f}^{(t)}\right\|_2^2$$

where the last inequality follows since the L1 norm is always bigger or equal to the L2 norm.

(3) The converse is not necessarily true. That is, a key could be very L2-heavy while being L1-heavy. For example, for the vector $f = (\sqrt{n}, 1, \ldots, 1)$ we have $f[1]^2 = n$ and $\|f\|_2^2 \approx 2n$. So key 1 is very L2-heavy. But it is not L1 heavy ($\sqrt{n}$ compared to $\approx n$).

(4) This means that, in spirit, aiming to identify all L2-heavy-hitters is a harder task than identifying all L1-heaby-hitters, since the L2-heavy-hitters as "less heavy" (using the same space).[1]

What is known about the L2-HH problem in the classical/oblivious setting? Can be solved with space $\boldsymbol{O(k \cdot \log n)}$ using the CountSketch algorithm (to be surveyed next)

What will we get from the generic construction for the adaptive setting based on DP-stability?
Recall our generic construction:

Input: Collection of $\boldsymbol{P}$ random strings $\boldsymbol{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_P}) \in (\{\mathbf{0},\mathbf{1}\}^*)^{\boldsymbol{P}}$ for some parameter $\boldsymbol{P}$

1. Initiate $\boldsymbol{P}$ copies on a non-adaptive algorithm $\boldsymbol{\mathcal{A}}$ with random strings $\boldsymbol{r_1}, \ldots, \boldsymbol{r_P}$
2. Every time an update comes, feed it to all of the copies of $\boldsymbol{\mathcal{A}}$
3. On query time:
   a) Query each of the copies of $\boldsymbol{\mathcal{A}}$ to obtain $\boldsymbol{P}$ intermediate outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_P}$
   b) Return a DP-stable aggregation of $\{\boldsymbol{y_1}, \ldots, \boldsymbol{y_P}\}$ as the current output

- Informally, composition arguments for DP-stability allow us to release $\boldsymbol{t}$ aggregations using a dataset of size $\boldsymbol{P} \approx \sqrt{\boldsymbol{t}}$
- In our context, we have $\boldsymbol{T}$ rounds, during each of which we need to release $\boldsymbol{O(k)}$ heavy elements
- This amounts to a total of $\boldsymbol{t = Tk}$ aggregations, for which we need $\boldsymbol{P} \approx \sqrt{\boldsymbol{Tk}}$ intermediate outputs
- This gives a total space of $\boldsymbol{O\left(\sqrt{T} \cdot k^{1.5}\right)}$. So the "price for robustness" is $\approx \sqrt{\boldsymbol{T \cdot k}}$

Can we do better?

[1] This is only "in spirit" as there could be keys which are L1 heavy but not L2 heavy (up to the difference between $\sqrt{k}$ and k mentioned above). For example, for $f = (10,90)$ we have that $f[1] \geq \frac{\|f\|_1}{10} = 10$, but $(f[1])^2 < \frac{\|f\|_2^2}{10} = \frac{100+8100}{10} = 820$. So key 1 is L1 heavy but not L2 heavy (both with parameter $k = 10$).

Algorithm CountSketch

- Parameters: n=dimension of input vector, d=size of sketch, b=$O(k)$=the "width" (where b/k controls "accuracy" as will become clear later)
- Initialize $\ell = \frac{d}{b}$ pairs of hash functions $\rho = \left((h_1, s_1), \ldots, (h_\ell, s_\ell)\right)$ where h_j: $[n] \to [b]$ and s_j: $[n] \to \{\pm 1\}$
- Think of ρ as defining $\frac{d}{b} \times b$ "buckets" $C[\cdot,\cdot]$ (initially empty)
- Given an update (i, Δ): For every $j \in [d/b]$ add $s_j(i) \cdot \Delta$ to the bucket $C[j, h_j(i)]$

Update (i, Δ)

$h_1(i)=5$, $s_1(i)=+1$					$+\Delta$				
$h_2(i)=2$, $s_2(i)=-1$		$-\Delta$							
$h_3(i)=8$, $s_3(i)=-1$								$-\Delta$	
$h_4(i)=3$, $s_4(i)=+1$			$+\Delta$						

Utility analysis:

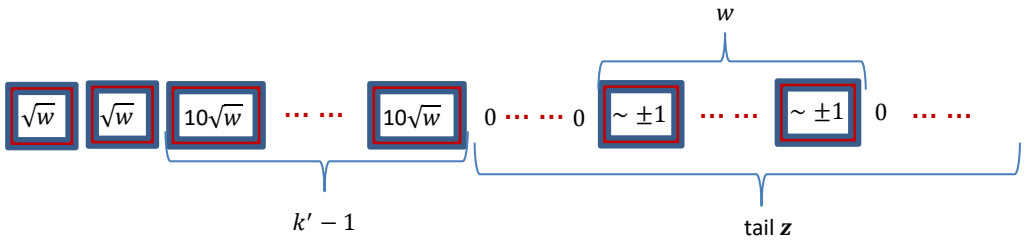
- For every $j \in [d/b]$ and $i \in [n]$ we have that $s_j(i) \cdot C[j, h_j(i)]$ is an unbiased estimator for $f[i]$
- Fix a bucket. Every key i is mapped to that bucket with value $f[i]$ or $-f[i]$ with probabilities $\frac{1}{2b}$
- Hence, the noise in the bucket is the sum of n "Bernoulli RVs" with variance $\approx \frac{1}{b}\sum_i (f[i])^2 = \frac{1}{b}\|f\|_2^2$
- If key i is an L2-heavy hitter then it overtakes the noise (since $|f[i]|$ is bigger than the standard deviation of the noise). Formally, fix a key i and consider its jth bucket $C[j, h_j(i)]$
  - By Chebyshev's inequality: $\mathbf{Pr}\left[\left|s_j(i)C[j, h_j(i)] - f[i]\right| > \sqrt{\frac{3}{b}} \cdot \|f\|_2\right] \le \frac{\frac{1}{b}\|f\|_2^2}{\left(\sqrt{\frac{3}{b}} \cdot \|f\|_2\right)^2} \le \frac{1}{3}$
  - That is, for every $j \in [d/b]$, with probability at least 2/3 we have that $s_j(i)C[j, h_j(i)]$ is an accurate estimate for $f[i]$ up to additive error of $\sqrt{\frac{3}{b}} \cdot \|f\|_2$.
  - Therefore, by the Chernoff bound (and a union bound over the n keys), if the number of rows $[d/b]$ is at least $\approx \log n$ then w.h.p. for every key i we have that $\mathbf{Median}_{j \in [d/b]}\left\{ s_j(i) \cdot C[j, h_j(i)] \right\}$ is within $\sqrt{\frac{3}{b}} \cdot \|f\|_2$ of $f[i]$.
- That is, from the sketch we can estimate the value of key i as $\mathbf{Median}_{j \in [d/b]}\left\{ s_j(i) \cdot C[j, h_j(i)] \right\}$
- Note that for an L2-heavy key i it holds that $\sqrt{\frac{3}{b}} \cdot \|f\|_2 < \sqrt{\frac{3k}{b}} \cdot |f[i]|$, which means multiplicative error at most $\left(1 + \sqrt{\frac{3k}{b}}\right)$ for heavy keys.
- To identify the list of heavy elements we report $O(k)$ keys with maximal estimated values:
  - To see that this list indeed includes all heavy elements, note that if key i is heavy with parameter k (i.e., $|f[i]| \ge \frac{1}{\sqrt{k}} \cdot \|f\|_2$) and key j is *not* heavy with parameter $2k$ (i.e., $|f[j]| <$

$\frac{1}{\sqrt{2k}} \cdot \|f\|_2$), then our estimation for $f[i]$ is bigger than our estimation for $f[j]$, because for all $b \geq 5k$ we have $\frac{1}{\sqrt{k}} \cdot \|f\|_2 - \sqrt{\frac{3}{b}} \cdot \|f\|_2 > \frac{1}{\sqrt{2k}} \cdot \|f\|_2 + \sqrt{\frac{3}{b}} \cdot \|f\|_2$

- That is, only keys that are at least "$2k$ heavy" might outshine keys that are "k heavy". As there could be at most $2k$ keys that are "$2k$ heavy", we get that all "k heavy" keys must be included in the list of $2k$ keys with top estimations.

Attack on CountSketch (median estimator)

- CountSketch reports exactly k' elements with highest estimated values
- Attack vectors are a concatenation of a fixed prefix of length $k'+1$ of "heavy" keys and a "tail"
- Tails have disjoint supports
- The reported candidate HH are $\mathrm{H} = \{1,3,\dots,k'+1\}$ **OR** $\{2,3,\dots,k'+1\}$

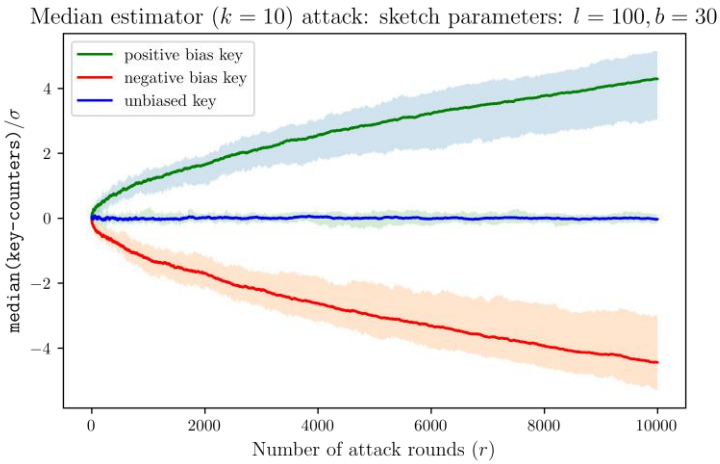


$a \leftarrow 0$
Repeat r times:
  Query with a random tail z
  If $1 \in H$ $a \leftarrow a + z$
  else $a \leftarrow a - z$
Return a

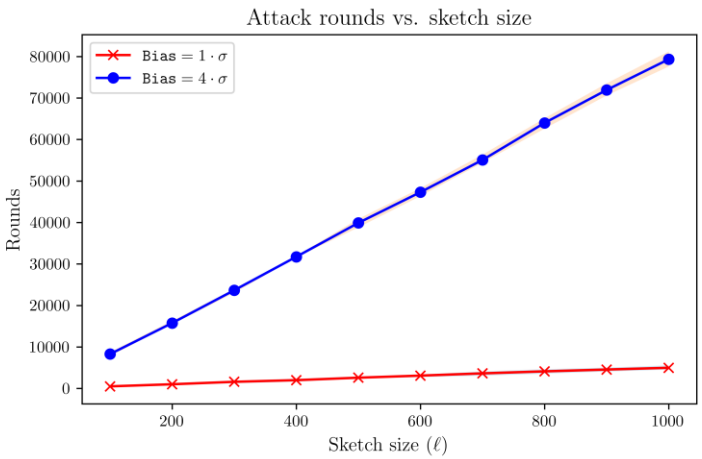
- a biases "towards" 1 and "away from" 2
- Adversarial vector: "0" on keys 1,2 and heavy on k' keys and "tail" a
- Keys 1,2 get large estimates and are reported in H

Attack Simulations

Simulations show that $r = 5 \cdot \tau^2 \cdot \frac{d}{b}$ attack queries suffice for generating an adversarial vector f with keys biased by $\frac{\tau}{\sqrt{k}} \cdot \|f\|_2$



Bias-to-noise ratio (τ) for number of rounds, average of 10 simulations with different initializations (shaded region between the minimum and maximum)



Attack rounds versus sketch size $\ell = d/b$ to obtain bias-to-noise ratio $\tau \in \{1,4\}$, averaged over 20 and 5 simulations respectively (shaded region between minimum and maximum)

Attack Analysis (simplified)

- Suppose for simplicity that we can get estimates for weights, and consider the following attack:

> $\boldsymbol{a \leftarrow 0}$
> **Repeat $\boldsymbol{r}$ times:**
> **Query value of 1 on a random tail $\boldsymbol{z}$ ($\boldsymbol{w}$ disjoint keys)**
> **If $\mathbf{Val}(1) < 0$ then $a \leftarrow a + z$**
> **Return $\boldsymbol{a}$**

- Estimates are computed as the median of $\boldsymbol{\ell = \frac{d}{b}}$ buckets in which 1 participates. The value of every such bucket is a "binomial-like" RV consisting of the sum of w "bits" which are 1 or -1 each wp $\frac{1}{2b}$, and 0 otherwise. The standard deviation of the value of every bucket is $\approx \sqrt{w/b}$.
- In expectation, 50% the ℓ buckets in which 1 participates are positive and 50% are negative. The standard deviation of the difference between number of positives and number of negatives is $\approx \sqrt{\ell} = \sqrt{d/b}$
- When we collect a tail, there are more negatives than positives buckets
- Whp, we collected $\boldsymbol{R = \Omega(r)}$ "good tails" in which there are $\sqrt{\ell}$ more negatives than positives
- Fix a bucket in which 1 participates, and sample a "good tail". The bucket is negative wp $\geq \frac{\frac{\ell}{2}+\sqrt{\ell}}{\ell} = \frac{1}{2} + \frac{1}{\sqrt{\ell}}$
- After R such samples, the bucket is negative in $\geq \frac{R}{2} + \frac{R}{\sqrt{\ell}} \pm \sqrt{R}$ and is positive in $\leq \frac{R}{2} - \frac{R}{\sqrt{\ell}} \pm \sqrt{R}$
- Assuming $R \gg \ell$ we get about $R/\sqrt{\ell}$ extra negative times, which gives bias

$$\approx \frac{R}{\sqrt{\ell}} \cdot \sqrt{\frac{w}{b}}$$

  % This is similar to the expectation of the sum of $\left(\ell - \frac{R}{\sqrt{\ell}}\right)$ unbiased buckets, which is zero, and the sum of $\frac{R}{\sqrt{\ell}}$ positive buckets, each of which is $\approx \sqrt{\frac{w}{b}}$

- We want this bias to overtake the random noise in the bucket, which is $\approx \sqrt{Rw/b}$, which happens when $R \gg \ell$

  % This is the standard deviation of the sum of Rw "bits" which are ± 1 wp $\frac{1}{2b}$ and 0 otherwise
- That is, we caused CountSketch to estimate $\mathrm{Val}(1) \gtrsim \sqrt{Rw/b}$ even though $\mathrm{Val}(1) = 0$. When $R \gg \ell$, then this is more than the allowed error level of $\frac{1}{\sqrt{k}}\sqrt{R \cdot w}$.

PS: We need $w = w(\ell)$. If w is constant then the green text is incorrect, because the probability of bucket being zero is $\frac{1}{\sqrt{w}}$ meaning that a constant fraction of the buckets are zero, and hence so is the median, and we get $\mathrm{Val}(1) = 0$ (in which case no tail is collected).

Warmup towards a robust construction

- Consider a single copy of CountSketch
- Instead of estimating values using the actual median of the buckets, return a DP-stable estimate of it
- So every time we return an estimate we aggregate $\frac{d}{b}$ buckets
- Thus, for releasing $O(T \cdot k)$ estimates overall we need to set $\frac{d}{b} \approx \sqrt{T \cdot k}$, so sketch size $\approx \frac{d}{b} \times b \approx \sqrt{T} \cdot k^{1.5}$
- So we didn't gain much in terms of the sketch size
- **Conceptual improvement:** In the generic construction, stability was w.r.t. a complete sketch. Now stability is w.r.t. one row, i.e. one pair of hash functions. Maintaining stability w.r.t. "smaller" objects is potentially easier.
- **Can we take this idea further?** Would it suffice to maintain stability w.r.t. a single bucket?

BCountSketch

- Recall that the generalization properties of DP-stability (as we have seen them in Theorem 8 in Lecture 6) hold when the data is sampled in an iid manner. However, In CountSketch, different buckets in the same row are slightly correlated.
- This is a minor issue and there is in fact a variant of CountSketch, called **BCountSketch**, where the buckets are iid:
  - For every bucket $j \in [d]$ we have a hash function $g_j: [n] \to \{-1, 0, 1\}$ where $\forall i \in [n]$

$$\Pr[g_j(i) = 0] = 1 - \frac{1}{b} \qquad \text{and} \qquad \Pr[g_j(i) = 1] = \Pr[g_j(i) = -1] = \frac{1}{2b}$$

  - Very similar performances (basically the same analysis)

Fine-tuned stability analysis

- Fix a key i and let B be a bucket that i participates in. As we have seen in the analysis of CountSketch (see page 3), in the oblivious case, with constant probability (say 99%) B provides an accurate estimation for $f[i]$ (up to the allowed error).
- Assuming that our algorithm maintains DP-stability w.r.t. the buckets, then by the generalization properties of DP-stability it follows that (whp) even in the adaptive case, 98% of the buckets in which i participates provide good estimates for it. Good!

- **How should we design a DP-stable algorithm w.r.t. the buckets?**
- Note that by working with the buckets as our "stability unit", we now have a dataset of size d instead of d/b as we had when working with the rows. Good!
  - % (But recall that every key participates in $\approx d/b$ buckets, so we need to use this big dataset carefully because when estimating its value we still only have d/b elements to aggregate…)

- Fix a set $\boldsymbol{H}$ of $\boldsymbol{k'}$ keys (think of them as the keys reported as heavy at some time step). If we were to resample a bucket $\boldsymbol{j} \in [\boldsymbol{d}]$ (i.e., sample $\boldsymbol{g_j}$), then in expectation this bucket contains $\frac{\boldsymbol{k'}}{\boldsymbol{b}} < \boldsymbol{1}$ keys from $\boldsymbol{H}$
- Similarly, fix a sequence of sets $\boldsymbol{H_1}, \dots, \boldsymbol{H_T}$, each containing $\boldsymbol{k'}$ (possibly different) keys. If we were to resample a bucket $\boldsymbol{j} \in [\boldsymbol{d}]$ (i.e., sample $\boldsymbol{g_j}$), then in expectation this bucket contains $\frac{\boldsymbol{k'}}{\boldsymbol{b}} < \boldsymbol{1}$ keys from each $\boldsymbol{H_r}$ and thus a total of $< \boldsymbol{T}$ keys from $\boldsymbol{H_1}, \dots, \boldsymbol{H_T}$
- By Markov, the probability that a bucket contributes to more than $\boldsymbol{100T}$ such keys is $< \frac{\boldsymbol{1}}{\boldsymbol{100}}$

- Suppose we delete a bucket once it contributes to $\boldsymbol{100T}$ reported keys. Then in the oblivious setting the expected number of deleted buckets is at most $\frac{\boldsymbol{d}}{\boldsymbol{100}}$
- A similar argument shows that for every fixture of a key $\boldsymbol{i}$, in the oblivious setting the expected number of buckets that are both deleted and contain $\boldsymbol{i}$ is at most $\approx \frac{\boldsymbol{d/b}}{\boldsymbol{100}}$
  - % To see this, note that the probability that a bucket $\boldsymbol{B}$ contains $\boldsymbol{i}$ and contains $\boldsymbol{100T}$ keys from $\boldsymbol{H_1}, \dots, \boldsymbol{H_T}$ is at most that probability that $\boldsymbol{B}$ contains $\boldsymbol{i}$ and contains $\boldsymbol{99T}$ keys different than $\boldsymbol{i}$ from $\boldsymbol{H_1}, \dots, \boldsymbol{H_T}$, which happens w.p. at most $\frac{\boldsymbol{1}}{\boldsymbol{b}} \cdot \frac{\boldsymbol{1}}{\boldsymbol{99}} \approx \frac{\boldsymbol{1}}{\boldsymbol{100b}}$
- Assuming that our algorithm maintains DP-stability w.r.t. the buckets, then again by the generalization properties of DP-stability it follows that (whp) even in the adaptive case, 98% of the buckets in which $\boldsymbol{i}$ participates are not deleted. Good!

- **Midway summary:** As long as we maintain DP-stability w.r.t. the buckets, then (w.h.p.) for every key $\boldsymbol{i}$ throughout all of the execution:
  (a) 98% of the buckets in which $\boldsymbol{i}$ participates provide good estimates for it
  (b) 98% of the buckets in which $\boldsymbol{i}$ participates are not deleted

- We next leverage these properties to present an algorithm based on **ThresholdMonitor** from Lecture 8. Recall that in **ThresholdMonitor**, if we delete an element after its $\boldsymbol{T}$'s contribution, then we need to add noise of magnitude $\approx \sqrt{\boldsymbol{T}}$ to our counts.
- Our counts will be of the form: "How many buckets estimate the value of a key X to be more than some threshold Y".
- As our noise if of the order of $\approx \sqrt{\boldsymbol{T}}$, we need to ensure than the number of "alive and accurate" buckets containing the key X, which is roughly $\boldsymbol{0.98 \cdot 0.98 \cdot d/b}$ is more than $\approx \sqrt{\boldsymbol{T}}$
- This will mean that the size of our (robust) sketch needs to satisfy $\boldsymbol{d} \gtrsim \sqrt{\boldsymbol{T}} \cdot \boldsymbol{b} \approx \sqrt{\boldsymbol{T}} \cdot \boldsymbol{k}$, instead of $\sqrt{\boldsymbol{T}} \cdot \boldsymbol{k^{1.5}}$ as we got from the generic construction

Algorithm Robust-BCountSketch

1. For every $\boldsymbol{j} \in [\boldsymbol{d}]$ sample a hash function $\boldsymbol{g_j}: [\boldsymbol{n}] \to \{-\mathbf{1}, \mathbf{0}, \mathbf{1}\}$ and initiate an empty bucket $\boldsymbol{c_j} = \mathbf{0}$

2. Initialize algorithm **ThresholdMonitor** on $(\boldsymbol{g_1}, \dots, \boldsymbol{g_d})$ with threshold $\approx \sqrt{\boldsymbol{T}}$ and deletion parameter $\approx \mathbf{100}\boldsymbol{T}$

3. When receiving an update $(\boldsymbol{i}, \boldsymbol{\Delta})$ where $\boldsymbol{i} \in [\boldsymbol{n}]$ is the key and $\boldsymbol{\Delta} \in \mathbb{R}$ is the weight:
   a. For every $\boldsymbol{j} \in [\boldsymbol{d}]$ do $\boldsymbol{c_j} \leftarrow \boldsymbol{c_j} + \boldsymbol{g_j}(\boldsymbol{i}) \cdot \boldsymbol{\Delta}$.
      (Note that most buckets are unchanged as $g_j(i) = 0$)

4. When queried (at most $\boldsymbol{T}$ queries):
   a. Set $\boldsymbol{T} = \emptyset$
   b. For $\boldsymbol{v} = \boldsymbol{m}, (\mathbf{1} - \boldsymbol{\alpha})\boldsymbol{m}, \dots, \mathbf{1}$ do: (here m bounds the length of the stream)
      i. For $\boldsymbol{i} \in [\boldsymbol{n}]$ do:
         - Define the counting query $f_{v,i}: [d] \to \{0,1\}$ as
           $$\boldsymbol{f_{v,i}(j) = 1} \quad \Leftrightarrow \quad \left|\boldsymbol{g_j(i) \cdot c_j}\right| \geq \boldsymbol{v}$$
           (That is, $\boldsymbol{f(j) = 1}$ if key $\boldsymbol{i}$ participates in the $\boldsymbol{j}$th bucket and the value of this bucket is at least $\boldsymbol{v}$ in absolute value)
         - Submit the query $\boldsymbol{f_{v,i}}$ to **ThresholdMonitor**. If the answer is ⊤ then add $\boldsymbol{i}$ to $\boldsymbol{H}$. If $|\boldsymbol{H}| = \boldsymbol{k'} = \boldsymbol{O}(\boldsymbol{k})$ then GOTO 4c
   c. Report $\boldsymbol{H}$