Lecture 7: Adaptive streaming (continued)

Source: Lecture notes by Aaron Roth and Adam Smith

Lecturer: Uri Stemmer

Deterministic algorithms

Theorem: Let A be a streaming model that works in the classical (=oblivious / non-adaptive) model. If A is deterministic, then it also works in the adaptive/adversarial model.

Proof sketch: Suppose not. Then there exists an adversary B that interacts with A and fails it in the adaptive model. That is, B is an algorithm that adaptively selects the updates to A such that A fails. But since A is deterministic, then the same input sequence that failed it in the adaptive setting must also fail it in the non-adaptive setting as A behaves exactly in the same manner.

Are there interesting deterministic streaming algorithms?

***The k-Heavy-Hitters Problem*:**

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

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

(Note: The list may also include non-heavy elements, which is fine, as long as it contains all the heavy ones.)

Theorem: There exists a deterministic streaming algorithm for this problem with space complexity $O\big(k \cdot (\log m + \log n)\big)$.

Proof: Consider the following algorithm

The Misra–Gries Algorithm

Initialization: Let $\boldsymbol{A}$ be an empty array

Step at stage $\boldsymbol{i}$ (for $\boldsymbol{1 \le i \le m}$):

1. Receive the current update $\boldsymbol{s_i}$
2. Update $\boldsymbol{A[s_i] \leftarrow A[s_i] + 1}$
3. If $\boldsymbol{|\{x \in [n] : A[x] > 0\}| = k}$ then:
   For all $\boldsymbol{x}$ where $\boldsymbol{A[x] > 0}$ do $\boldsymbol{A[s_i] \leftarrow A[s_i] - 1}$
4. Return the list of elements $\boldsymbol{\{x \in [n] : A[x] > 0\}}$

Algorithm Analysis – Memory: Note that at each step of the execution, there are at most k non-zero entries in the array. It suffices to store the indices of these entries (k numbers in the range [n]) and the values of the array at these positions (each value is an integer between 1 and m). Therefore, the total memory is $O\big(k \cdot (\log m + \log n)\big)$.

Algorithm Analysis – Correctness:

Let i be any stage in the execution, and let $x^* \in [n]$ be a heavy element, i.e., $\text{weight}_i(x^*) > \frac{i}{k}$. Note that every time x^* appears in the stream, the value $A[x^*]$ increases by 1. Thus, to show that x^* is returned in step 4 at the end of stage i, it suffices to show that the value of $A[x^*]$ decreases (by 1) at most i/k times throughout the execution.

To see why this is true, consider the following equivalent formulation of the algorithm. In this equivalent formulation, we track not only the counter's value but also the set of indices contributing to that counter. (This would require more memory, but we use this version only for the proof.)

Equivalent Formulation of the Algorithm (a conceptual algorithm for the proof)

Initialization: Let B be an empty array

Step at stage i (for $1 \le i \le m$):

1. *Receive the current update s_i*
2. *Update $B[s_i] \leftarrow B[s_i] \cup \{i\}$*
3. If $|\{x \in [n] : B[x] \neq \emptyset\}| = k$ Then:
   a. For all x where $B[x] \neq \emptyset$ do $: B[s_i] \leftarrow B[s_i] \setminus \min(B[s_i])$
      (i.e., we forget the index of the earliest occurrence of s_i)
4. *Return the list of elements $\{x \in [n] : B[x] \neq \emptyset\}$*

This algorithm is equivalent to the previous algorithm because, at any point during the execution, for every x it holds that $A[x] = |B[x]|$.

We want to show that the value $A[x^*]$ decreases at most $\frac{i}{k}$ times in the execution of the first algorithm, or equivalently, that the set $B[x^*]$ shrinks at most $\frac{i}{k}$ times in the execution of the second algorithm.

Now, note that every time the set $B[x^*]$ shrinks, exactly k sets shrink, and the indices removed from these k sets are distinct in the sense that no index can be removed more than once throughout the execution.

This means that if the set $B[x^*]$ shrinks T times during the execution, there are Tk distinct indices that are removed during these shrinking times. Since the stream (so far) has length i, it must hold that $T \leq i/k$.

q.e.d.

Conclusion: For the k-Heavy-Hitters problem, there exists a streaming algorithm in the adaptive model with $O\big(k \cdot (\log m + \log n)\big)$ memory.

Sketch Switching

Now we will show a transformation from the classical model to the adaptive model, which works under the assumption that the objective function cannot "jump" too many times during the execution.

Example: Suppose we are interested in estimating the number of distinct elements in a stream of length m, and assume there are no deletions in the stream. This is a monotonically increasing function whose value is bounded by m. How many times during the execution can the value of this function jump by a multiplicative factor of $(1+\alpha)$?

$$(1+\alpha)^t \leq m$$

$$t \cdot \underbrace{\log(1+\alpha)}_{\approx \alpha} \leq \log m$$

$$t \lesssim \frac{1}{\alpha} \cdot \log m$$

More generally: Suppose we have a bound λ on the number of times the objective function can "jump" (increase or decrease) by a multiplicative factor of $(1+\alpha)$ during the execution, and assume that λ is not too large. We can use this to design an adaptive algorithm:

Assume we have an algorithm $\boldsymbol{\mathcal{A}}$ that operates in the non-adaptive world. Specifically, in the non-adaptive world, algorithm $\boldsymbol{\mathcal{A}}$ guarantees that with probability at least $1-\beta$, all answers it returns during the execution are α-accurate.

Consider the following algorithm:

Algorithm Switch

1. Initiate $\boldsymbol{\lambda}$ independent instances of the oblivious algorithm $\boldsymbol{\mathcal{A}}$, denoted as $\boldsymbol{\mathcal{A}_1, \ldots, \mathcal{A}_\lambda}$
2. Let $\boldsymbol{r = 1}$ and let $\mathbf{OUT} = \mathbf{0}$
3. For $\boldsymbol{i = 1, 2, \ldots, m}$:
   a) Receive next update $(\boldsymbol{u_i, \Delta_i})$ and feed it to all copies of algorithm $\boldsymbol{\mathcal{A}}$
   b) Let $\boldsymbol{y_i}$ be the answer returned by $\boldsymbol{\mathcal{A}_r}$
   c) If $\mathbf{OUT} \notin (\mathbf{1} \pm \boldsymbol{\alpha}) \cdot \boldsymbol{y_i}$ then set $\mathbf{OUT} \leftarrow \boldsymbol{y_i}$ and set $\boldsymbol{r \leftarrow r + 1}$ (if $\boldsymbol{r > \lambda}$ then fail)
   d) Output $\mathbf{OUT}$

Now we want to show that as long as the algorithm does not fail, then with high probability, the answers it returns are accurate (even in the adaptive model).

For the analysis, fix $\ell \in [\lambda]$, and consider the following algorithm:

Algorithm Switch-$\boldsymbol{\ell}$

1. Initiate $\boldsymbol{\lambda}$ independent instances of the oblivious algorithm $\boldsymbol{\mathcal{A}}$, denoted as $\boldsymbol{\mathcal{A}_1, \ldots, \mathcal{A}_\lambda}$
2. Let $\boldsymbol{r = 1}$ and let $\mathbf{OUT} = \mathbf{0}$
3. For $\boldsymbol{i = 1, 2, \ldots, m}$:
   a) Receive next update $(\boldsymbol{u_i, \Delta_i})$ and feed it to all copies of algorithm $\boldsymbol{\mathcal{A}}$
   b) Let $\boldsymbol{y_i}$ be the answer returned by $\boldsymbol{\mathcal{A}_r}$
   c) If $(\boldsymbol{r < \ell})$ and $\mathbf{OUT} \notin (\mathbf{1} \pm \boldsymbol{\alpha}) \cdot \boldsymbol{y_i}$ then set $\mathbf{OUT} \leftarrow \boldsymbol{y_i}$ and set $\boldsymbol{r \leftarrow r + 1}$
   d) Output $\mathbf{OUT}$

Claim: With probability at least $1 - \beta$, all the answers returned by $\boldsymbol{\mathcal{A}_\ell}$ during the execution of ***Switch-$\boldsymbol{\ell}$*** are $\boldsymbol{\alpha}$-accurate.

Explanation: In algorithm ***Switch-$\boldsymbol{\ell}$***, we are not actually using $\boldsymbol{\mathcal{A}_\ell}$. Therefore, the sequence of inputs is independent of its randomness. This means that this copy of the algorithm operates in the non-adaptive world...

Notation: For $\ell \in [\lambda]$, let i_ℓ denote the index of the iteration where r takes the value $\ell + 1$ for the first time

- In algorithm ***Switch***, this is the time when we first reveal an output obtained from the copy $\boldsymbol{\mathcal{A}_\ell}$. In algorithm ***Switch-$\boldsymbol{\ell}$***, this is the time when we *would* have revealed it, but did not do so due to the red condition in the algorithm

Claim: Fix an adaptive adversary, assume without loss of generality that it is deterministic, and also fix the randomness of the copies of algorithm $\mathcal{A}$. Then, up to (and including) time i_ℓ, the sequence of outputs returned by $\mathcal{A}_\ell$ during the execution of ***Switch-ℓ*** is identical to the sequence of outputs returned during the execution of ***Switch***.

Explanation: The changes we made in ***Switch-ℓ*** only start to affect the execution with the output returned in iteration i_ℓ. The entire prefix of the execution remains unchanged. In particular, the input received at this iteration does not change. Therefore, the current output of all the copies we maintain (including $\mathcal{A}_\ell$) also remains identical.

Conclusion: With probability at least $1 - \beta$, up to (and including) time i_ℓ, the sequence of outputs returned by $\mathcal{A}_\ell$ during the execution of ***Switch*** are α-accurate.

However, note that algorithm ***Switch*** does not use the copy $\mathcal{A}_\ell$ after time i_ℓ.

Conclusion: With probability at least $1 - \lambda\beta$, all intermediate outputs used by algorithm **Switch** are α-accurate.

Explanation: This follows from the union bound over all choices of $\ell \in [\lambda]$.

Final Conclusion: With probability at least $1 - \lambda\beta$, algorithm **Switch** is $O(\alpha)$-accurate even in the adaptive model.

Transformation using DP-Stability

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\widetilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

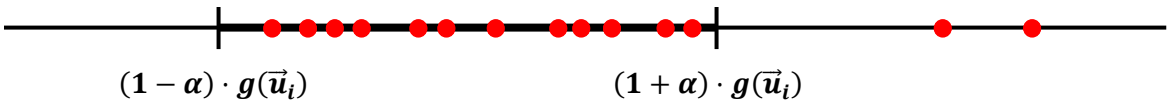
Algorithm $\mathcal{B}$
Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \ldots, r_k) \in (\{0,1\}^*)^k$ 2. Initiate k independent instances $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \ldots, r_k$ 3. For $i = 1, 2, \ldots, m$: a) Receive next update (u_i, Δ_i) b) Insert update (u_i, Δ_i) into each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \ldots, y_{i,k}$ c) Output $z_i = \text{DP\_Stable\_Median}(y_{i,1}, \ldots, y_{i,k})$ using stability parameter $\varepsilon_0 \approx 1/\sqrt{m}$

Analysis idea:

- $\mathcal{B}$ is DP-stable w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = \left((u_1, \Delta_1), \ldots, (u_i, \Delta_i)\right)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a DP-stable computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP-stability we have

$$\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\{y_{i,j} \text{ is accurate}\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r\left[f_{\vec{u}_i}(r)\right] \approx 1$$

- So, most of the $y_{i,j}$'s are accurate, and hence, any approximate median is also accurate



Discussion:
On one hand, inflating the memory by a factor of $\sqrt{m}$ is non-trivial, and specifically, it is a *sublinear* inflation (so if the original algorithm uses significantly less memory than $\sqrt{m}$, we obtain an adaptive algorithm with sublinear memory in m).

On the other hand, $\sqrt{m}$ is significant compared to the memory bounds we typically aim for in the non-adaptive streaming world, where we usually target memory on the order of $\text{polylog}(m)$. This raises the question: Is this avoidable in general?

Separation Between Adaptive and Non-Adaptive Streaming

Definition: Let X be a domain and let $\vec{u} = \left((x_1, b_1), \ldots, (x_\ell, b_\ell)\right) \in (X \times \{\pm 1\})^*$ be a sequence of updates (or stream), where each update contains an element from X and a binary weight. For an element $x \in X$ denote

$$weight(\vec{u}, x) = \sum_{i: x_i = x} b_i \neq 0$$

And denote:

$$count(\vec{u}) = |\{x : weight(\vec{u}, x) \neq 0\}|$$

In other words, $count(\vec{u})$ counts the number of elements for which the total weight received in the stream is currently non-zero.

Definition: In the *OneFromMany* problem, the input is a stream of the form described above. If $count(\vec{u}) < |X|/100$, then any output is valid. Otherwise, the algorithm must return an element $x \in X$ such that $weight(\vec{u}, x) \neq 0$.

Claim: In the **non-adaptive model**, there exists an algorithm for the *OneFromMany* problem with constant failure probability (as small as desired) and memory $\text{polylog}(m, |X|)$.

Explanation: At the start of the execution, the algorithm samples a subset $A \subseteq X$ of size $\text{polylog}\, m$. During the stream, the algorithm tracks (exactly) the weights of all elements in A. At each step, the algorithm returns an element $a \in A$ with a non-zero weight, if such an element exists; otherwise, it returns an arbitrary element from X.

To see why this works, note the following:

- By the problem definition, the algorithm cannot fail at times t when fewer than $|X|/100$ elements have a non-zero weight.
- Now fix a time t where more than $|X|/100$ elements have a non-zero weight. By Chernoff's bound, with high probability, the sampled set A contains at least one such element. In this case, the algorithm succeeds.

The claim then follows by applying a union bound over all time steps.

q.e.d.

Theorem: In the **adaptive model**, any algorithm for the **OneFromMany** problem that fails with probability at most ¾ requires **linear memory** in $|X|$ (assuming the stream is sufficiently long).

To prove this claim, we will use the following combinatorial statement:

Combinatorial Claim: There exists a set $B \subseteq 2^X$ with the following properties:

1. For every $Y \in B$ it holds that $|Y| = |X|/2$
2. For every $Y, Z \in B$ it holds that $|Y \cap Z| < \frac{49|X|}{100}$
3. $|B| \geq \frac{2^{|X|/5}}{\sqrt{2|X|}}$

Proof of the Combinatorial Claim:

We build B using the following greedy algorithm:

1. Initiate $B = \emptyset$ and $R = \{Y \subseteq X : |Y| = |X|/2\}$.
2. While $R \neq \emptyset$
   (a) Let $Y \in R$
   (b) Set $B \leftarrow B \cup \{Y\}$
   (c) Remove from R every set Z such that $|Y \cap Z| \geq \frac{49|X|}{100}$
4. Return B

By construction, the returned set B satisfies conditions 1 and 2 of the claim. As for condition 3, note that every iteration deletes at most

$$\binom{|X|/2}{49|X|/100} \cdot \binom{51|X|/100}{|X|/100} \leq \left(\frac{100e}{98}\right)^{49|X|/100} \cdot (51e)^{|X|/100} < 2^{0.8|X|}$$

sets Z from R. To see this, note that once Y is fixed, we could bound the number of choices for Z as follows. There are $\binom{|X|/2}{49|X|/100}$ ways to choose $49|X|/100$ elements from Y to be in Z. After that, there are at most $\binom{51|X|/100}{|X|/100}$ ways to select $|X|/100$ additional elements to Z (in order to make it of size $|X|/2$) from the remaining $51|X|/100$ elements.

Recall that, initially, in Step 1 of the greedy algorithm, we have that $|R| = \binom{|X|}{|X|/2} \geq 2^{|X|}/\sqrt{2|X|}$, so the number of iterations of the greedy algorithm (and hence the size of B) is at least

$$\frac{2^{|X|}/\sqrt{2|X|}}{2^{0.8|X|}} = \frac{2^{|X|/5}}{\sqrt{2|X|}}$$

q.e.d.

Proof of the theorem:
Assume, for contradiction, that there exists an algorithm $\mathcal{A}$ that solves the *OneFromMany* problem in the adaptive model with success probability at lease ¼. Let B be the set guaranteed by the combinatorial claim. Consider the following thought experiment:

Input: $Y \in B$
1. For every $x \in Y$, feed algorithm $\mathcal{A}$ the update $(x, 1)$
2. Initiate $\hat{Y} = \emptyset$
3. Repeat $49|X|/100$ times:
   a. Obtain an outcome $x \in X$ from $\mathcal{A}$
   b. Add x to $\hat{Y}$
   c. Feed the update $(x, -1)$ to $\mathcal{A}$
4. In there is an element $Z \in B$ such that $\hat{Y} \subseteq Z$ then return Z. Otherwise return $\perp$

Note that if algorithm $\mathcal{A}$ does not fail, then after step 3 we get that $\hat{Y}$ is a subset of Y of size $\frac{49|X|}{100}$.

Since any two sets in B agree on fewer than $\frac{49|X|}{100}$ elements, $\hat{Y}$ cannot be a subset of any set in B other than Y. Therefore, if algorithm $\mathcal{A}$ succeeds, the output of the thought experiment is $Z = Y$. In this case, we say that the thought experiment succeeded.

By our assumption on $\mathcal{A}$, for any input Y, the thought experiment succeeds with probability at least ¼.

Thus, there exists a fixing of the randomness of $\mathcal{A}$ for which the thought experiment succeeds for at least $|B|/4$ of the possible inputs Y from B.

Otherwise, if we sample Y uniformly from B, we get that

$$\frac{1}{4} \leq \Pr_{r,Y}[\mathcal{A}_r(Y) \text{ succeeds}] = \sum_r \Pr[r] \cdot \Pr_Y[\mathcal{A}_r(Y) \text{ succeeds}] < \sum_r \Pr[r] \cdot \frac{1}{4} = \frac{1}{4}$$

which is a contradiction… here r denotes the randomness of $\mathcal{A}$

That is, after fixing the randomness of $\mathcal{A}$, there exists a subset $B_0 \subseteq B$ of size $|B_0| \geq \frac{|B|}{4}$, such that for every $Y \in B_0$, if we run our thought experiment on Y, the experiment will succeed, and the output will be $Y = Z$.

Now note that the internal state of algorithm $\mathcal{A}$ at the end of step 1 determines the output of our thought experiment. Therefore, since there are at least $\frac{|B|}{4}$ distinct outputs for the thought experiment, then algorithm $\mathcal{A}$ must have at least $\frac{|B|}{4}$ distinct internal states, and thus its memory must be at least:

$$\log\left(\frac{|B|}{4}\right) = \Omega(|X|)$$