15.1.2025 ***Adaptive data analysis***

Lecture 10: Negative Results

Source: Lecture notes by Aaron Roth and Adam Smith

Lecturer: Uri Stemmer

In our course, we saw several methods that allow answering adaptive statistical queries. In particular, we saw:

- A computationally efficient algorithm that answers k adaptive queries using a sample of size
$$n \gtrsim \sqrt{k}$$
- A computationally inefficient algorithm that answers k adaptive queries using a sample of size
$$n \gtrsim (\log k)^3 \cdot (\log|X|)^2$$

Is there a computationally efficient algorithm that achieves significantly better results than the algorithm we saw? Today we will see that, in general, the answer is no.

The idea: We will present an attacker (analyst) who interacts with an efficient mechanism that answers queries. Using (approximately) n^2 queries, our attacker will succeed in reconstructing almost the entire sample held by the mechanism. As we have already seen in our course, once the attacker reconstructs the sample, he can easily find a "bad" query.

For this, we will need to become familiar with the following cryptographic tool:

Fingerprinting codes (FPC)

This is a cryptographic tool designed to try and prevent the illegal distribution of digital content. Suppose Netflix releases a new episode of a certain series to its subscribers every day. Netflix wants to prevent malicious users from downloading the episode and uploading it to the internet. To attempt to prevent this, they can embed invisible watermarks in the episodes they release, so that each subscriber receives a unique copy with different markings. Now, if Netflix finds the episode online, they can identify which subscriber distributed it.

But what happens if several subscribers decide to collaborate, compare their copies to try and remove the markings, and then upload a "processed" copy to the internet?

This problem is what FPC aims to solve. The idea relies on the following assumption:

- If two malicious users attempt to combine their copies to remove the markings, they will only be able to identify locations where their markings differ. However, if at a certain location they both have the exact same marking, they won't notice there is a marking there and will not remove it.

- Similarly, if a group of n malicious users tries to collaborate and remove the markings from their copies, they will not be able to identify markings that are common to all of them. That is, if at a certain point in the file they all have the exact same marking, that marking will remain in the "processed" copy they try to create.

Under these assumptions, FPC enables Netflix to identify the malicious users.

In more detail:

- There are $\boldsymbol{N}$ users, out of which $\boldsymbol{n}$ users have decided to collaborate and upload episodes to the internet.
- Each day:
  - Netflix broadcasts a new episode with different markings for each user.
  - The $\boldsymbol{n}$ malicious users collaborate to try to remove or alter the markings in their copies, under the restriction that markings identical across all $\boldsymbol{n}$ malicious users cannot be removed. They then upload the processed copy they created to the internet.
    - Formally, we will (seemingly) require even less from the malicious users: markings that are identical across all $\boldsymbol{N}$ users cannot be removed. (In particular, if markings identical across the $\boldsymbol{n}$ malicious users cannot be removed, then markings identical across all $\boldsymbol{N}$ users clearly cannot be removed either.)
- Netflix observes the processed episode on the internet.
- After $\boldsymbol{\ell}$ such iterations, Netflix can identify (almost) all of the malicious users.

Formally, we consider the following game between an adversary $\boldsymbol{\mathcal{P}}$, who controls $\boldsymbol{n}$ out of $\boldsymbol{N}$ users, and an algorithm $\boldsymbol{\mathcal{F}}$ (this is the FPC algorithm), which attempts to identify the $\boldsymbol{n}$ users controlled by the adversary.

Algorithm $\boldsymbol{IFPC(\mathcal{P},\mathcal{F})}$

(1) Denote $\mathrm{ALIVE}^1 = [N]$
(2) The Adversary $\mathcal{P}$ *selects a subset of users* $S^1 \subseteq [N]$ *(unknown to* $\mathcal{F}$*)*
(3) For $j = 1$ *to* ℓ *do*
  a. *Algorithm* $\mathcal{F}$ *outputs a vector* $c^j \in \{-1,1\}^N$
  b. *Let* $c^j_{S^j}$ *be the restriction of* c^j *to coordinates in* S^j*. This is given to* $\mathcal{P}$
  c. *The adversary* $\mathcal{P}$ *outputs* $a^j \in \{-1,1\}$*. This is given to* $\mathcal{F}$
  d. *Algorithm* $\mathcal{F}$ *<u>accuses</u> a set of users* $I^j \subseteq [N]$*.*
  e. *Let* $\mathrm{ALIVE}^{j+1} = \mathrm{ALIVE}^j \setminus I^j$*, and let* $S^{j+1} = S^j \setminus I^j$
(4) The Adversary $\mathcal{P}$ *<u>wins</u> if it was <u>consistent</u> throughout the game. Formally,* $\mathcal{P}$ *is consistent if for every* $j \in [\ell]$ *there exists an index* $i \in \mathrm{ALIVE}^j$ *such that* $a^j = c^j_i$

Definition 1: Algorithm $\mathcal{F}$ is an FPC with parameters n, N, ℓ, ε if for any adversary $\mathcal{P}$, the following conditions hold:

(1) $\Pr\left[\begin{array}{c}\text{A good user is} \\ \text{accused as malicious}\end{array}\right] \leq \varepsilon$

(2) $\Pr[\text{the adversary } \mathcal{P} \text{ wins}] = \Pr\left[\begin{array}{c}\mathcal{P} \text{ is consistant throughout} \\ \text{all of the execution}\end{array}\right] \leq \varepsilon$

where the probabilities are over the execution of the game *IFPC*

What's happening here?
Throughout the execution, we accuse users of being malicious, and they are "removed from the game" (in particular, they are removed from the set S^j). We require that, with high probability, no honest user is accused of being malicious.

Additionally, we require that at some point during the execution, because malicious users are progressively removed, the adversary $\mathcal{P}$ will no longer be able to satisfy the consistency requirement. Since we assume that attackers cannot produce "processed episodes" without meeting the consistency requirement, this means the attackers will no longer be able to produce processed episodes.

Theorem 2: For every $1 \leq n \leq N$, there exists an FPC with parameters n, N, ℓ, ε, where $\varepsilon = \frac{1}{N}$ and $\ell = O(n^2 \cdot \log N)$

Partial Proof of Theorem 2:
We provide a partial proof of the theorem, using the parameter $\ell = \mathbf{poly}(n)$ instead of $\ell \approx n^2$.

Our current goal is to construct an algorithm $\mathcal{F}$ for the aforementioned IFPC game. Recall that, intuitively, during the game, algorithm $\mathcal{F}$ attempts to "accuse" more and more malicious users, until (hopefully) it accuses all of the malicious users.

We will begin by designing an algorithm (denoted $\mathcal{F}_1$) whose goal is to accuse a single malicious user from the set of malicious users. Then, we will iteratively use this algorithm to identify additional malicious users.

Algorithm $\mathcal{F}_1$

(1) Let Z be the following matrix:

$$\begin{pmatrix} \textbf{0 block} & \textbf{1st block} & \textbf{2dn block} & & \boldsymbol{n}\textbf{th block} \\ 000\ldots000 & 111\ldots111 & 111\ldots111 & & 111\ldots111 \\ 000\ldots000 & 000\ldots000 & 111\ldots111 & \cdots & 111\ldots111 \\ 000\ldots000 & 000\ldots000 & 000\ldots000 & & 111\ldots111 \\ \vdots & \vdots & \vdots & & 111\ldots111 \\ 000\ldots000 & 000\ldots000 & 000\ldots000 & & 111\ldots111 \end{pmatrix}$$

In this matrix, there are $(n+1)$ types of columns, with $\tilde{O}(n^2)$ copies of each type of column. Therefore, the total number of columns in this matrix is $\tilde{O}(n^3)$.

(2) Let Y be a matrix obtained by applying a random permutation to the columns of matrix Z.

(3) For $\ell = \tilde{O}(n^3)$ rounds, algorithm $\mathcal{F}_1$ outputs (one after the other) the columns of the matrix Y as the watermark vectors it is supposed to output in the IFPC game

(4) Let $\vec{a} \in \{0,1\}^{\ell}$ denote the vector of responses that algorithm $\mathcal{F}_1$ received from the adversary during step (3.

(5) For each column type/block $\boldsymbol{i}$, let $\mathbf{count}_{\boldsymbol{i}}$ denote number of columns of type $\boldsymbol{i}$ for which the adversary returned the response $\mathbf{1}$. Note that this is a number between $\mathbf{0}$ and $\widetilde{\boldsymbol{O}}(\boldsymbol{n^2})$

(6) Return an index $\boldsymbol{i} \in [\boldsymbol{n}]$ (i.e., the accused user) for which the following holds:

$$\mathbf{count}_{\boldsymbol{i}} > \mathbf{count}_{\boldsymbol{i-1}} + \widetilde{\boldsymbol{O}}(\boldsymbol{n})$$

First, we will show that step (6) is well-defined, meaning that an index $\boldsymbol{i}$ as described indeed exists. Recall that we assume the adversary must be consistent (otherwise, we win, and we don't care). Thus:

- For all columns of type $\mathbf{0}$, the adversary must return the response $\mathbf{0}$.
- For all columns of type $\boldsymbol{n}$, the adversary must return the response $\mathbf{1}$.

This implies that:

$$\mathbf{count}_{\mathbf{0}} = \mathbf{0}$$
$$\mathbf{count}_{\boldsymbol{n}} = \widetilde{\boldsymbol{O}}(\boldsymbol{n^2})$$

Therefore, since there are $\boldsymbol{n}+\mathbf{1}$ types of columns, there must be an index $\boldsymbol{i}$ that satisfies the condition from step (6) in the algorithm.

Next, we will show that, with high probability, an index $\boldsymbol{i}$ that satisfies this condition must correspond to a "malicious" user. To this end, let $\boldsymbol{i}$ be an index of an "honest" user. We will show that the probability of $\boldsymbol{i}$ satisfying the condition is negligible.

To see this, note that since user $\boldsymbol{i}$ is "honest," then the adversary cannot distinguish between columns from block $(\boldsymbol{i-1})$ and columns from block $\boldsymbol{i}$, because the only row that differentiates them is row $\boldsymbol{i}$, and the adversary never sees the entries of this row.

Thus, intuitively, the adversary's responses must behave similarly across these two blocks, and there cannot be a "jump" of $\approx \boldsymbol{n}$ in the counts.

More formally: Fix the adversary's responses and the entire permutation, except for the partition of the columns (after the permutation) between those belonging to block $(\boldsymbol{i-1})$ and those belonging to block $\boldsymbol{i}$. Since the adversary does not see row $\boldsymbol{i}$, this partition between the columns of $(\boldsymbol{i-1})$ and $\boldsymbol{i}$ remains completely random, even given all the adversary's responses.

Let $\vec{\boldsymbol{a}}_{\boldsymbol{i}}$ denote the set of the adversary's responses corresponding to the columns from blocks $(\boldsymbol{i-1})$ and $\boldsymbol{i}$. Now the random variables $\mathbf{count}_{\boldsymbol{i-1}}$ and $\mathbf{count}_{\boldsymbol{i}}$ have the same marginal distribution: each of them is the sums of a randomly chosen half of the bits in $\vec{\boldsymbol{a}}_{\boldsymbol{i}}$. Therefore, these two variables have exactly the same expectation. Furthermore, by the Hoeffding bound, with overwhelming probability, the sums will not deviate by more than $\tilde{\boldsymbol{O}}(\boldsymbol{n})$ from their expectation (since each sum involves $\tilde{\boldsymbol{O}}(\boldsymbol{n^2})$ bits). Hence, the probability of a large "jump" occurring for any "honest" user $\boldsymbol{i}$ is negligible.

Where do we stand in the proof of Theorem 2?

We have shown that using $\tilde{O}(n^3)$ rounds, we can accuse one "malicious" user. We then run this algorithm n times. Each run is conducted only on the set of users who are still "alive", and in each iteration we remove one user from the set of "alive" users. In total, after $\tilde{\boldsymbol{O}}(\boldsymbol{n^4})$ rounds, we will identify all the malicious users (or the adversary will "lose" somewhere along the way, meaning they will fail to maintain consistency, in which case we also win).

Q.E.D. (Theorem 2)

Now, let us see how FPC can be used to derive negative results for the problem of answering adaptively chosen statistical queries. Specifically, we will show that efficient mechanisms cannot answer more than $\boldsymbol{n^2}$ queries using a sample of size $\boldsymbol{n}$.

Simplifying assumption: The query-answering mechanism satisfies the following property:

Definition: A query-answering mechanism is called "natural" if, when it holds a sample $\boldsymbol{S} \in \boldsymbol{X^n}$ and receives a query $\boldsymbol{q}: \boldsymbol{X} \to \{\pm\mathbf{1}\}$, its output does not depend on the values of the query outside the sample. That is, for any two queries $\boldsymbol{q}$ and $\boldsymbol{q'}$ that agree on all the elements of the sample $\boldsymbol{S}$, the answer returned by the algorithm is identical (or distributed identically).

Theorem:
Let $\boldsymbol{M}$ be a natural mechanism that answers $\boldsymbol{k}$ adaptive queries over a domain of size $|\boldsymbol{X}| \geq \mathbf{2000}\boldsymbol{n}$ using a sample of size $\boldsymbol{n}$. Then, $\boldsymbol{n} = \boldsymbol{\Omega}\left(\sqrt{\boldsymbol{k}}\right)$.

Proof Sketch:
Consider the following attack. The adversary defines a target distribution $\mathfrak{D}$, from which the mechanism $\boldsymbol{M}$ receives a sample (the adversary does not see the sample). The adversary then asks (approximately) $\boldsymbol{n^2}$ adaptive queries. For at least one of these queries, the mechanism will fail to answer accurately.

Attack against a natural mechanism $\boldsymbol{M}$

Setup:

- *Given parameter $\boldsymbol{n}$, let $\boldsymbol{N} = \mathbf{2000}\boldsymbol{n}$ and let $\mathfrak{D}$ be the uniform distribution over $\{\mathbf{1}, \dots, \boldsymbol{N}\}$*
- *Let $\boldsymbol{S}$ be a sample containing $\boldsymbol{n}$ iid elements from $\mathfrak{D}$. The sample $\boldsymbol{S}$ is given to the mechanism $\boldsymbol{M}$ (but not to the adversary)*

The Attack:

1. *Initialize an $(\boldsymbol{n}, \boldsymbol{N}, \boldsymbol{\ell}, \boldsymbol{\varepsilon})$-FPC algorithm $\mathcal{F}$, where $\boldsymbol{\ell} = \widetilde{\boldsymbol{O}}\left(\boldsymbol{n^2}\right)$ and $\boldsymbol{\varepsilon} \leq \frac{\mathbf{1}}{\boldsymbol{N}}$*
2. *Let $\boldsymbol{T^0} = \emptyset$*
3. *For $\boldsymbol{j} = \mathbf{1}$ to $\boldsymbol{\ell}$ do*
   a. *Let $\boldsymbol{c^j} \in \{\pm\mathbf{1}\}^{\boldsymbol{N}}$ be the vector chosen by $\mathcal{F}$*
   b. *Define the query $\widehat{\boldsymbol{q}}^{\boldsymbol{j}}$ such that $\widehat{\boldsymbol{q}}^{\boldsymbol{j}}(\boldsymbol{i}) = \begin{cases} \boldsymbol{c_i^j} & , \ \textbf{if } \boldsymbol{i} \notin \boldsymbol{T^{j-1}} \\ \mathbf{0} & , \ \textbf{else} \end{cases}$*
   c. *Ask $\widehat{\boldsymbol{q}}^{\boldsymbol{j}}$ to the mechanism $\boldsymbol{M}$ and obtain answer $\boldsymbol{a^j}$. Round $\boldsymbol{a^j}$ to $\overline{\boldsymbol{a}}^{\boldsymbol{j}} \in \{\pm\mathbf{1}\}$*
   d. *Give $\overline{\boldsymbol{a}}^{\boldsymbol{j}}$ to $\mathcal{F}$ and let $\boldsymbol{I^j} \subseteq [\boldsymbol{N}]$ be the set of accused users. Let $\boldsymbol{T^j} \leftarrow \boldsymbol{T^{j-1}} \cup \boldsymbol{I^j}$*

First, note that by to the properties of the FPC, with probability at least $\mathbf{1} - \frac{\mathbf{1}}{\boldsymbol{N}}$, in every round $\boldsymbol{j}$, it holds that $\boldsymbol{I^j} \subseteq \boldsymbol{S}$, since we assume that false accusations occur with probability at most $\frac{\mathbf{1}}{\boldsymbol{N}}$.

In this case, for every $\boldsymbol{j}$, it holds that $\left|\boldsymbol{T^j}\right| \leq \boldsymbol{n}$. Let us assume this is indeed the case

Additionally, according to the guarantees of FPC, with probability at least $\mathbf{1} - \frac{\mathbf{1}}{\boldsymbol{N}}$, there exists a round $\boldsymbol{j}$ where the response $\overline{\boldsymbol{a}}^{\boldsymbol{j}}$ is inconsistent. In particular, this means that at time $\boldsymbol{j}$, the vector $\boldsymbol{c^j}$

must be either identically **-1** or identically **1** (since if this vector contains both **-1**'s and **1**'s, then by definition, $\overline{\boldsymbol{a}}^{\boldsymbol{j}}$ cannot be inconsistent.)

This means that at time $\boldsymbol{j}$, one of two cases occurs:

Case 1: $\overline{\boldsymbol{a}}^{\boldsymbol{j}} = -\mathbf{1}$ but $\boldsymbol{c}^{\boldsymbol{j}} \equiv \mathbf{1}$.

Case 2: $\overline{\boldsymbol{a}}^{\boldsymbol{j}} = \mathbf{1}$ but $\boldsymbol{c}^{\boldsymbol{j}} \equiv -\mathbf{1}$.

In either case, this implies that the response returned by the mechanism is very far from the expected value of the query in that round. Specifically, in Case 1,

$$\hat{\boldsymbol{q}}^{\boldsymbol{j}}(\mathfrak{D}) \geq \mathbf{1} - \frac{\left|\boldsymbol{T}^{\boldsymbol{j}}\right|}{\boldsymbol{N}} \geq \mathbf{1} - \frac{\boldsymbol{n}}{\boldsymbol{N}} \geq \mathbf{1} - \frac{\mathbf{1}}{\mathbf{2000}}$$

And in Case 2,

$$\hat{\boldsymbol{q}}^{\boldsymbol{j}}(\mathfrak{D}) \leq -\mathbf{1} + \frac{\left|\boldsymbol{T}^{\boldsymbol{j}}\right|}{\boldsymbol{N}} \leq -\mathbf{1} + \frac{\boldsymbol{n}}{\boldsymbol{N}} \leq -\mathbf{1} + \frac{\mathbf{1}}{\mathbf{2000}}$$

This shows that the rounded answer $\overline{\boldsymbol{a}}^{\boldsymbol{j}}$ is very far from the "true answer" $\hat{\boldsymbol{q}}^{\boldsymbol{j}}(\mathfrak{D})$, to the extent that the unrounded answer (the answer provided by the mechanism $\boldsymbol{M}$) is also very inaccurate.

q.e.d.

Question: Where did we use the simplifying assumption that the mechanism is "natural"?

Answer: The guarantees of FPC hold only under the rules of the IFPC game. An important point is that in the IFPC game, the adversary $\boldsymbol{\mathcal{P}}$ only observes the values $\boldsymbol{c}^{\boldsymbol{j}}_{\boldsymbol{S}^{\boldsymbol{j}}}$, i.e., only the coordinates of the vector $\boldsymbol{c}^{\boldsymbol{j}}$ corresponding to the elements in $\boldsymbol{S}$ that have not yet been accused as malicious.

In contrast, in the last attack we presented, we give the mechanism $\boldsymbol{M}$ the entire vector $\boldsymbol{c}^{\boldsymbol{j}}$ (this vector defines the statistical query we present to the mechanism). In general, this falls outside the rules of the IFPC game and breaks its guarantees. However, under our simplifying assumption—that the mechanism is "natural"—we know that it "ignores" the values of this query outside the sample, i.e., it "does not look" at the coordinates of $\boldsymbol{c}^{\boldsymbol{j}}$ it is not supposed to see. Hence, the attack works.

Note: We did not make any assumptions here about the computational power of the mechanism. In other words, what we have shown is that a query-answering mechanism, if it is "natural," cannot answer more than $\boldsymbol{n}^{\mathbf{2}}$ queries—even if it is not computationally limited.

Note: To eliminate this simplifying assumption, we can use encryption to hide from the mechanism the content of the query in the coordinates it is not supposed to see. This effectively forces the mechanism to behave like a "natural" mechanism, and the attack still works. (However, in this case, we would need to assume that the mechanism is computationally limited, as otherwise, it could break the encryption and access the hidden coordinates.)

Negative result for adaptive streaming

Let us see how to use the above impossibility results to also derive a negative result for **adaptive streaming**. The difference from the impossibility result we previously saw for adaptive streaming is that now we will obtain a negative result for a streaming problem defined by a real-valued function (it is going to be a negative result for an "estimation" problem rather than a "search" problem).

The Plan:
We will present a streaming problem that can be easily solved in the **non-adaptive** model but cannot be solved with small space in the **adaptive** model. Specifically, we will show that if this problem can be solved effectively in the adaptive model, it would imply the existence of an efficient mechanism that answers adaptive queries too effectively, which we know is impossible.

The SADA Problem:

- Each update s_i in the stream is either a point $s_i = p_i \in X$ or a function $s_i = h_i : X \to \{0,1\}$.
- **Goal:** At each step i, after receiving the update s_i, return an approximation of the average of the most recent function received over the multiset containing all the points received in the stream so far.

Formally:

- Let $S = (s_1, s_2, \dots, s_i)$ be a prefix of the stream.
- Let f_S denote the most recent function appearing in S.
- Let $P_S = \{s_j : j \in [i] \ \& \ s_j \in X\}$ denote the multiset of all points appearing in S.
- Define the target function as:

$$g(S) = \frac{1}{|P_S|} \sum_{x \in P_S} f_S(x)$$

Intuitively, in the **non-adaptive world**, the SADA problem is easy to solve because at each step, we can estimate the average of f_S using a small random sample from P_S. That is, we do not need to store the entire multiset P_S (which could be very large) to approximate the target function.

But, in the **adaptive world**, we know that a small sample would not suffice in order to answer many adaptive queries. So, if we want to approximate average of f_S using a sample from P_S, then intuitively our sample size must be large which requires large space.

Theorem (Informal): There exists a computationally efficient algorithm for the SADA problem in the non-adaptive model that uses $\mathbf{O}(\mathbf{polylog}(m))$ memory, where m is the length of the stream.

Proof Sketch:
All we need to do is understand how a streaming algorithm can maintain a random sample of the points that have appeared in the stream. Once we know how to do this, we can answer the queries using the sample, and achieve accuracy (in the non-adaptive model) using Chernoff bound.

Moreover, it suffices to show a streaming algorithm that maintains **a single random sample** throughout the stream, because if we can do this, we can simply run y independent instances of the algorithm in parallel to obtain a sample of size y.

Reservoir Sampling [Vitter, 1985]:

1. Init $\boldsymbol{s} = \perp$
2. When the next item u_i is read, toss a biased coin and with probability $\boldsymbol{1/i}$ let $\boldsymbol{s = u_i}$ (note that we need to maintain both the element $\boldsymbol{s}$ and a counter for $\boldsymbol{i}$)
3. Output $\boldsymbol{s}$

Theorem: Assume that the elements in the stream come from a domain of size $\boldsymbol{n}$, specifically $\boldsymbol{u_i} \in [\boldsymbol{n}]$, and that the stream has length $\boldsymbol{m}$. Then, the above algorithm uses $\boldsymbol{O(\log(n) + \log(m))}$ memory, and its output is an element uniformly sampled from the stream, i.e., for every $\boldsymbol{i} \in [\boldsymbol{m}]$ we have $\Pr[s = u_i] = \frac{1}{m}$.

Proof: By induction on $\boldsymbol{i} \in [\boldsymbol{m}]$

Assume the theorem holds after the first $\boldsymbol{j}$ steps. That is, after $\boldsymbol{j}$ steps, for every $\boldsymbol{i} \in [\boldsymbol{j}]$, it holds that

$$\boldsymbol{\Pr[s = u_i] = \frac{1}{j}}$$

We will show that the property holds after the next step:

- By construction, for $\boldsymbol{u_{j+1}}$, it holds that $\boldsymbol{\Pr[s = u_{j+1}] = \frac{1}{j+1}}$
- And for $\boldsymbol{i} \in [\boldsymbol{j}]$ we have

$$\boldsymbol{\Pr[s = u_i] = \Pr\left[\substack{s = u_i \\ \text{after } j \text{ steps}}\right] \cdot \Pr[u_i \text{ survives}] = \frac{1}{j} \cdot \left(1 - \frac{1}{j+1}\right) = \frac{1}{j} \cdot \frac{j}{j+1} = \frac{1}{j+1}}$$

q.e.d.

In contrast, as we will now see, the problem becomes difficult in the adaptive model:

Theorem (Informal): Any computationally efficient algorithm for the SADA problem in the adaptive model must use memory $\boldsymbol{\Omega(\text{poly}(m))}$.

The difficulty arises from the inability to efficiently answer many adaptive queries using a small sample. As we have seen, if we want to answer many adaptive queries, our sample size must be large, and therefore, intuitively, the memory required to solve the SADA problem must also be large.

Formally we will prove this via reduction: If there exists an adaptive streaming algorithm that solves the SADA problem "too well", we could construct from it a mechanism that answers adaptive queries "too well", which is impossible.

So let us assume that we have an algorithm $\mathcal{A}$ that solves the SADA problem in the adaptive world. To simplify the proof, assume that $\mathcal{A}$ is deterministic. We will construct from $\mathcal{A}$ a mechanism $\boldsymbol{M}$ that answers queries as follows:

Algorithm $\mathcal{M}$

Input: Dataset $\boldsymbol{P} = (\boldsymbol{p_1}, \dots, \boldsymbol{p_n}) \in \boldsymbol{X^n}$ containing $\boldsymbol{n}$ elements from the domain $\boldsymbol{X}$
Algorithm used: An adversarially robust streaming algorithm $\mathcal{A}$ for the SADA problem

1. For $\boldsymbol{p} \in \boldsymbol{P}$ feed the update $\boldsymbol{p}$ to $\mathcal{A}$
2. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{k}$:
    a) Obtain next function $\boldsymbol{f_i}: \boldsymbol{X} \to \{\boldsymbol{0}, \boldsymbol{1}\}$
    b) Feed the update $\boldsymbol{f_i}$ to $\mathcal{A}$ and obtain an answer $\boldsymbol{z_i}$
    c) Output $\boldsymbol{z_i}$

Let us try and analyze this algorithm:

- Fix a distribution $\mathfrak{D}$ over $\boldsymbol{X}$ and consider the execution of $\mathcal{M}$ on a sample $\boldsymbol{P}$ containing i.i.d. samples from $\mathfrak{D}$.
- By construction, if algorithm $\mathcal{A}$ succeeds in solving the SADA problem, then at every step $\boldsymbol{i}$ we have that
$$z_i \approx \frac{1}{|P|} \sum_{p \in P} f_i(p)$$
- This is good, but it's not exactly what we want... We want to ensure that $\mathcal{M}$'s answers are accurate with respect to the distribution $\mathfrak{D}$, not just the empirical average over $\boldsymbol{P}$!

Discussion: How do we complete the proof?