**CSCI 1951-W Sublinear Algorithms for Big Data** **Fall 2020**

# Lecture 4: List monotonicity in Hamming and $\ell_1$ distance

*Lecturer: Jasper Lee* *Scribe: Junewoo Park*

## 1 Few Points to Clarify from Last Class

1. In the distribution of the normalized sample mean $\frac{\sqrt{n}}{\sigma}\overline{X}_n$, there is a region (within $O(1)$ from its mean) that behaves like Gaussian. Meanwhile, we essentially don't know what is happening with tail, but the Central Limit Theorem claims that the sample mean behaves asymptotically like a Gaussian distribution. These two facts do not contradict each other since as we take $n \to \infty$, the "body" expands. However, the rate at which body increases can be arbitrarily slow.

2. Catoni's sample complexity lower bound is only applicable to the sample mean. Hence, it doesn't contradict the upper bound given by Median of Means method in the last lecture.

3. The Median of Means algorithm does not require knowing $\epsilon$ to run the algorithm. It can just split samples into $\Theta(\log \frac{1}{\delta})$ groups, sample means for each group, and take the median of the means. In fact, $\epsilon$ depends on $n$, $\delta$, and $\sigma$, and note that the variance is a-priori unknown.

## 2 List monotonicity (sortedness)

In this lecture, we will learn how list monotonicity can be tested.

Recall the framework of property testing. Given a set of objects $\mathcal{C}$ and a distance function $d : \mathcal{C} \times \mathcal{C} \to [0, 1]$ on $\mathcal{C}$, property is defined as a subset of $\mathcal{P} \subseteq \mathcal{C}$, and property testing for $\mathcal{P}$ is a decision problem to check whether a given object $\mathcal{O} \in \mathcal{C}$ is a member of $\mathcal{P}$ or "$\epsilon$-far" from $\mathcal{P}$.

In this lecture, $\mathcal{C}$ is the set of all lists of length $n$ and the property $M \subseteq \mathcal{C}$ in question is the subset of all non-decreasing lists. We consider two distance functions:

1. Hamming distance, defined as $\text{Hamming}(f_1, f_2) = \frac{1}{n}|\{\, i \in [n] : \; f_1(i) \neq f_2(i) \,\}|$

2. $\ell_1$ distance, defined as $\|f_1 - f_2\|_1 = \frac{1}{n} \sum_{i \in [n]} |f_1(i) - f_2(i)|$.

One obvious algorithm to check monotonicity is to pick $i$ randomly and check if $f(i) \leq f(i+1)$. This algorithm needs $\Theta(n)$ queries since on input

$$11 \cdots 100 \cdots 0$$

($\frac{n}{2}$ 1's and $\frac{n}{2}$ 0's), since a single query catches a (1, 0) pair only with probability $\frac{1}{n}$.

Another algorithm is to pick $i, j$ such that $i < j$ randomly and check $f(i) \leq f(j)$. The query complexity of such algorithm is also $\Theta(n)$, since on input

$$4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9, \cdots$$

the probability of catching a decreasing pair is

$$\mathbb{P}(i, j \text{ are in the same “group”}) \leq \max_i \mathbb{P}(j \text{ is in the same group as } i \big| i) \leq \frac{3}{n-1}$$

We will show that there is a testing algorithm with query/time complexity $O(\frac{1}{\epsilon}\log n)$. Also, it is known that there is an $\Omega(\log n)$ lower bound for constant $\epsilon$.

## 3 Monotonicity testing: Hamming distance

This section introduces a monotonicity testing and analyzes its performance with respect to the Hamming distance. Since $\log n$ complexity is associated with binary search, we can try coming up with an algorithm that incorporates binary search $O(\frac{1}{\epsilon})$ times. Assume for now that each list $f$ has distinct elements.

**Algorithm 4.1:** Monotonicity testing in Hamming distance

```
For $O(\frac{1}{\epsilon})$ many times:
    1. Pick random index $i$.
    2. Do a binary search for $f(i)$.
    3. If there are inconsistencies, such as start $<$ end or $f(i) \neq \text{found}(i)$, then reject.
If $f$ is not rejected yet, accept.
```

Indeed, Algorithm 4.1 has query/time complexity $O(\frac{1}{\epsilon}\log n)$. The following two results show its correctness.

**Proposition 4.2** (Completeness) *Algorithm 4.1 accepts all monotonic lists.*

**Proposition 4.3** (Soundness) *Algorithm 4.1 rejects a list $\epsilon$-far (with respect to the Hamming distance) from $M$ with probability at least $\frac{2}{3}$.*

*Proof.* Completeness is straightforward since sorted lists cannot have such errors.

For Soundness, suppose that the given list $f$ is not sorted. Say that an index is *good* if test does not reject for $i$. It suffices to show that if a single iteration in the test accepts with probability $> 1 - \epsilon$, then $f$ is less than $\epsilon$-far from $M$. In other words, we may show that if $1 - \epsilon$ fraction of indices are good, then $f$ cannot be $\epsilon$-far from $M$. This follows from the following lemma. □

**Lemma** Good indices form a monotonic sublist.

*Proof.* Suppose that $i < j$ are good indices. Let $K$ be the last index in both $i$ and $j$'s binary search path. Then, $i \leq k \leq j$. By the definition of good indices, $f(i) \leq f(k) \leq f(j)$, so monotonicity holds. □

Distinction assumption can be removed by reducing a general instance $f$ to a distinct instance $f'$, where $f'$ is defined as $f'(i) = (f(i), i)$. Just run Algorithm 4.1 on $f'$ with respect to the lexicographic ordering.

**Remark.** Note that Algorithm 4.1 is “proximity-oblivious” tester, which means in particular that it does not have to know the $\epsilon$-gap to run the algorithm.

Last remark on Algorithm 4.1 is that it is an adaptive algorithm.

**Definition 4.4** (Adaptivity, informal definition) An algorithm is *adaptive* if some of its query locations depend on other query results. It is called *non-adaptive* if it can commit to all query locations at the beginning.

**Remark.** Although Algorithm 4.1 is an adaptive algorithm, there is an analogous non-adaptive version; after the random indices are chosen, it is possible to precompute all indices on binary search paths to the chosen indices before querying them.

It turns out that there is a better bound for the special case of binary lists.

**Fact 4.5** (Homework 2) *For $\{0,1\}$-lists there is a non-adaptive tester with query/time complexity $\text{poly}(\frac{1}{\epsilon})$ (independent of $n$) for monotonicity.*

## 4 Monotonicity testing: $\ell_1$ distances

On lists on $[0,1]$, the difference between $f_1(i)$ and $f_2(i)$ contributes to $\frac{1}{n}$ in Hamming distance, but contributes to $\frac{|f_1-f_2|}{n} \leq \frac{1}{n}$ in $\ell_1$ distance. Hence, Hamming distance dominates $\ell_1$ distance.

**Proposition.** $\text{Hamming}(f_1, f_2) \geq \|f_1 - f_2\|_1$. Similarly, $\text{Hamming}(f, M) \geq d_{\ell_1}(f, M)$ holds. Meanwhile, the two distances are equal on binary lists.

**Corollary.** By the above proposition, since $\ell_1$ distance lower bounds Hamming distance, Algorithm 4.1 is also sound and complete for the monotonicity testing problem in $\ell_1$ distance, with query complexity $O(\frac{1}{\epsilon}\log n)$.

Another way to reduce $\ell_1$-distance testing is to use Fact 4.5 and reduce a $[0,1]$ instance to a binary instance. To do so, we start with the notion of thresholding as follows:

**Definition 4.6** (Thresholding) For a $[0,1]$-list $f$, define $f_{(t)}(i) = \mathbb{1}\{f(i) \geq t\}$ for $t \in [0,1]$. Then, it follows that $f(i) = \int_0^1 f_{(t)}(i)dt$

The thresholding functions contain all necessary information about the monotonicity of the given list $f$.

**Proposition 4.7** *$f$ is monotone if and only if $f_{(t)}$ is monotone for all $t$.*

**Algorithm 4.8:** Monotonicity testing: $\ell_1$ distance

```
1. Non-adaptively sample $\text{poly}(\frac{1}{\epsilon})$ indices as in $\{0,1\}^n$ tester from Fact 4.5.
2. Check monotonicity. Accept if the list is monotonic. Reject otherwise.
```

Again, Algorithm 4.8 is complete. To show soundness, we should relate $f$'s $\epsilon$-farness from $M$ (with respect to $\ell_1$) to $f_{(t)}$'s $\epsilon$-farness from $M$. This can be done using the following lemma.

**Lemma 4.9** *$d_{\ell_1}(f, M) = \int_0^1 d_{\ell_1}(f_(t), M)dt$ and $d_{\ell_1}(f, M) = \int_0^1 \text{Hamming}(f_{(t)}, M)dt$*

We now use Lemma 4.9 to prove Theorem 4.10, the soundness of Algorithm 4.8.

**Theorem 4.10** *Algorithm 4.8 rejects a list $\epsilon$-far from $M$ in $\ell_1$ sense with probability $\frac{2}{3}$.*

*Proof of Theorem 4.10.* Let $f$ be a list $\epsilon$-far from $M$. By Lemma 4.9 and the averaging argument, there exists $t$ such that $f_{(t)}$ is $\epsilon$-far from monotone. Thus, by Fact 4.5, with probability at least $\frac{2}{3}$, the non-adaptive query location include indices $i < j$ such that $f_{(t)}(i) > f_{(t)}(j)$. Since $f_{(t)}$ is a boolean list, $f_{(t)}(i) = 1$ and $f_{(t)}(j) = 0$. Hence, $f(i) \geq t > f(j)$ by the definition of thresholding. $\square$

*Proof of Lemma 4.9.* The second equation follows from Definition 4.6. It remains to prove the first equation.

First, we prove that $d_{\ell_1}(f, M) \leq \int_0^1 d_{\ell_1}(f(t), M)dt$. Let $g_t$ be the closest monotone function to $f_{(t)}$ and let $g(i) = \int_0^1 g_t(i)dt$. Integration preserves monotonicity, so $g$ is monotone. Thus,

$$\begin{aligned} d_{\ell_1}(f, M) \leq \|f - g\|_1 &= \left\| \int_0^1 (f_{(t)} - g_t)dt \right\|_1 \\ &\leq \int_0^1 \left\|f_{(t)} - g_t\right\|_1 dt = \int_0^1 d_{\ell_1}(f_{(t)}, M)dt \end{aligned}$$

where the inequality holds by triangle inequality and the last equality holds by definition.

Now, let $g$ be the closest monotone list to $f$. Then,

$$\begin{aligned} \int_0^1 d_{\ell_1}(f_{(t)}, M)dt &\leq \int_0^1 \left\|f_{(t)} - g_{(t)}\right\|_1 dt && \text{(by the definition of } d_{\ell_1}(f_{(t)}, M)) \\ &= \int_0^1 \frac{1}{n} \sum_i |f_{(t)}(i) - g_{(t)}(i)|dt \\ &= \frac{1}{n} \sum_i \int_0^1 |f_{(t)}(i) - g_{(t)}(i)|dt \\ &\overset{(*)}{=} \frac{1}{n} \sum_i \left| \int_0^1 f_{(t)}(i) - g_{(t)}(i)dt \right| \\ &= \left\| \int_0^1 (f_{(t)} - g_{(t)})dt \right\|_1 \\ &= \|f - g\|_1 && \text{(by the integration in Definition 4.6)} \\ &= d_{\ell_1}(f, M) \end{aligned}$$

where the third equality $(*)$ holds since for fixed $i$, the sign of $f_{(t)}(i) - g_{(t)}(i)$ does not change as $t$ varies. $\square$