COMS 6998-3: Sub-Linear Algorithms in Learning and Testing **Spring 2014**

Lecture 1: 01/22/2014

Lecturer: Rocco Servedio *Scribes: Clément Canonne and Richard Stark*

1 Today

- High-level overview
- Administrative details
- Some content

Relevant Readings:

- Ergün, Kannan, Kumar, Rubinfeld and Viswanathan, 1998: *Spot-checkers.* [EKK+98]
- Ron, 2008: *Property Testing: A Learning Theory Perspective.* [Ron08]

2 High-level overview

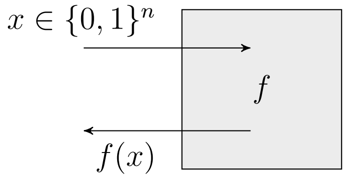
2.1 What is this course about?

Goal Get information from some *massive* data object – so humongous we cannot possibly look at the whole object. Instead, we must use *sublinear*-time algorithms to have any hope of getting something done.

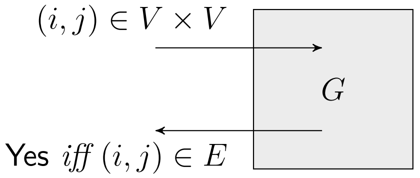
This immediately triggers the first natural question – is it even possible to do *anything*? As we shall see, the answer is – perhaps surprisingly – yes.

What kind of "objects"? We will be mainly interested in 3 different sorts of ginormous objects: Boolean functions, graphs and probability distributions.

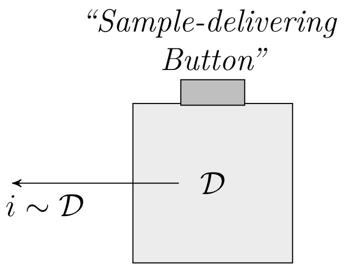
Example 1. *The object is a* Boolean function $f\colon \{0,1\}^n \to \{0,1\}$*, of size* 2^n*, to which we have* query access*:*



Example 2. *The object is a* graph $G = (V, E)$ *where* $V = [N] = \{1, \dots, N\}$*, for* $N = 2^n$*; we have query access to its adjacency matrix:*



Example 3. *The object is a* probability distribution $\mathcal{D}$ *over* $[N]$*, and we have access to independent samples:*



What can we hope for? For most questions, it is impossible to get an *exact* answer in sublinear time. Think for instance of deciding whether a function is identically zero or not; until all 2^n points have been queried, there is no way to be certain of the answer. However, by accepting *approximate* answers, we can do a lot.

Similarly, it is not hard to see that it is paramount that our algorithms are randomized. Deterministic ones are easy to "fool". So, we will seek algorithms that give good approximations with high probability.

$$\boxed{\text{randomization} + \text{approximation}}$$

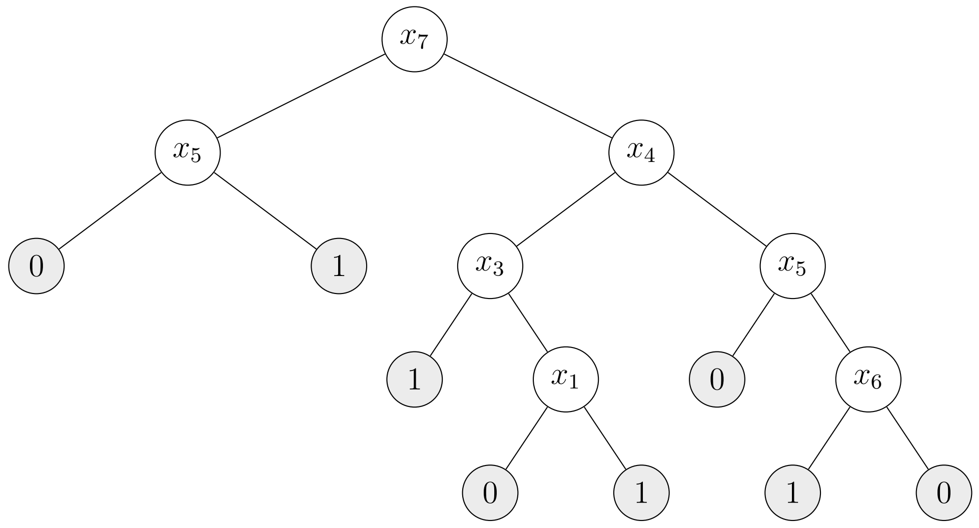
2.2 Kinds of algorithmic problems

We will consider two main families of problems over the three types of objects introduced above: learning and property testing.

2.2.1 Learning problems

The goal is to output some **high-quality approximation** of the object $\mathcal{O}$. Note that we can only do this if we know $\mathcal{O}$ is highly *structured*. For instance, learning a completely random Boolean function f defined by tossing a coin to generate a truth table (for each $x \in \{0,1\}^n$, $f(x)$ is chosen uniformly, independently at random in $\{0,1\}$) clearly cannot be done with an efficient number of queries.

Example 4 (Learning decision trees)**.** *Assume* $f\colon \{0,1\}^n \to \{0,1\}$ *is computed by a* $\mathrm{poly}(n)$*-size decision tree (DT). E.g., for* $x = (x_1, \dots, x_8)$*,* $f(x)$ *is given by the value at the leaf reached by going down the following tree (0: left, 1: right):*



(in this example, $f(11001100) = 1$*). The distance measure used here will be the* Hamming distance*: for* $f, g\colon \{0,1\}^n \to \{0,1\}$*,*

$$\mathrm{d}(f,g) \overset{\text{def}}{=} \frac{|\{\, x \in \{0,1\}^n \ :\ f(x) \neq g(x) \,\}|}{2^n} = \Pr_{x \sim \mathcal{U}_{\{0,1\}^n}}[\, f(x) \neq g(x) \,] \tag{1}$$

Question: *Can we, given black-box access to* f *(promised to be computed by a* $\mathrm{poly}(n)$*-size DT), run in* $\mathrm{poly}(n, 1/\epsilon)$ *time and output some hypothesis* $h\colon \{0,1\}^n \to \{0,1\}$ *s.t.* $\mathrm{d}(f,g) \leq \epsilon$*?*

$\triangleright$ *Spoiler: yes – we will cover this in future lectures.*

Example 5 (Learning distributions)**.** *Distribution $\mathcal{D}$ over $[N]$. Assume $\mathcal{D}$ has structure;*[1] *more particularly, for this example, assume $\mathcal{D}$ is* monotone *(non-increasing):*

$$\mathcal{D}(1) \geq \mathcal{D}(2) \geq \cdots \geq \mathcal{D}(N)$$

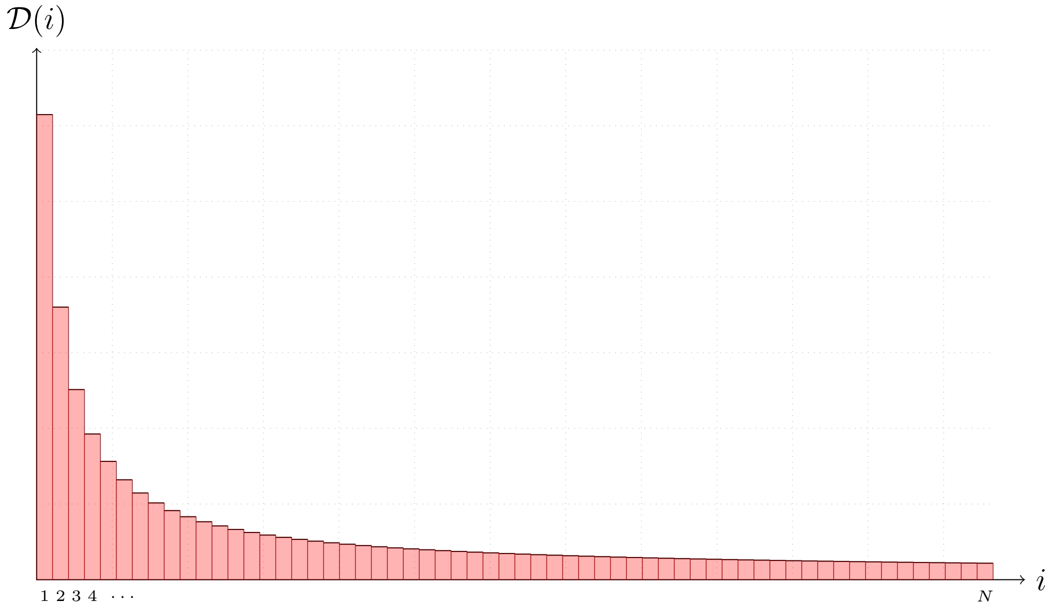


Figure 1: Example of a monotone distribution $\mathcal{D}$.

The distance measure considered will be the total variation distance *(TV)*[2]*: for $\mathcal{D}_1, \mathcal{D}_2$ distributions over $[N]$,*

$$\mathrm{d}_{\mathrm{TV}}(\mathcal{D}_1, \mathcal{D}_2) \overset{\text{def}}{=} \max_{S \subseteq [N]} (\mathcal{D}_1(S) - \mathcal{D}_2(S)) = \frac{1}{2} \sum_{i \in [N]} |\mathcal{D}_1(i) - \mathcal{D}_2(i)| \tag{2}$$

where the second equality (known as Scheffé's Identity) is left as an exercise. It is not hard to see that $\mathrm{d}_{\mathrm{TV}}(\mathcal{D}_1, \mathcal{D}_2) \in [0,1]$ (where the upper bound is for instance achieved for $\mathcal{D}_1$, $\mathcal{D}_2$ with disjoint supports).

Hint: can be shown by considering the set $S = \{ i \in [N] : \mathcal{D}_1(i) > \mathcal{D}_2(i) \}$.

[1] As we will show later in the class, if $\mathcal{D}$ is arbitrary, the number of samples are needed for learning is linear in N, specifically, $\Theta(N/\epsilon^2)$.

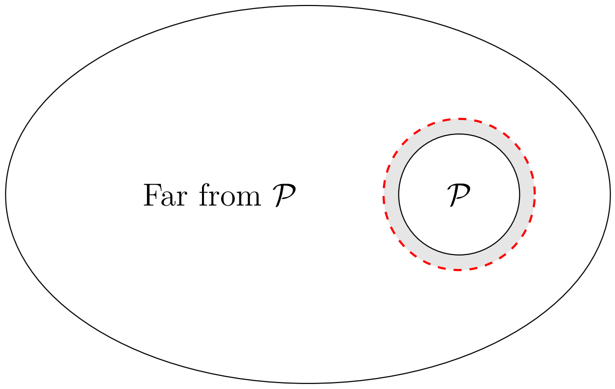
[2] A (very stringent) metric also referred to as *statistical distance*, or (half) the L_1 distance.

Question: *Given access to independent samples from some $\mathcal{D}$, unknown monotone distribution over $[N]$, what are the sample and time complexity required to output (a succinct representation of) a hypothesis distribution $\mathcal{D}'$ s.t. $\mathrm{d}_{\mathrm{TV}}(\mathcal{D}, \mathcal{D}') \leq \epsilon$?*

$\triangleright$ *Good news: can do this with $O(\log N/\epsilon^3)$ samples and runtime – and this is optimal.*

2.2.2 Property testing problems

The object $\mathcal{O}$ is now arbitrary, and we are interested in some *property* $\mathcal{P}$ on the set of all objects. The goal is to distinguish whether: (a) $\mathcal{O}$ has the property $\mathcal{P}$, of (b) $\mathcal{O}$ is "far" from every $\mathcal{O}'$ with property $\mathcal{P}$. We don't care about the in-between cases where $\mathcal{O}$ is "close" to having property $\mathcal{P}$. In such cases we can give whatever answer we want. Equivalently, this setting can be seen as a "promise" problem, where we are "promised" that all objects will either have property $\mathcal{P}$ or be "far" from having property $\mathcal{P}$.



For instance, for the property $\mathcal{P}$ (on Boolean functions) of "being the identically zero function", f is ϵ-far from $\mathcal{P}$ if it takes value 1 on at least an ϵ fraction of the inputs.

Example 6 (Testing $f\colon \{0,1\}^n \to \{0,1\}$ for monotonicity)**.**

Definition 7. *A Boolean function f is* monotone *(non-decreasing) if $x \preceq y$ implies $f(x) \leq f(y)$; where $x \preceq y$ means $x_i \leq y_i$ for all $i \in [n]$. For instance, f defined by $f(x) = x_1 \wedge x_{17}$ is monotone; $f(x) = \bar{x}_1$ is not.*

Taking our property $\mathcal{P}$ to be monotonicity (equivalently, $\mathcal{P} = \{\, f\colon \{0,1\}^n \to \{0,1\} \,:\, f \text{ is monotone} \,\}$; the set of all functions with the property), we define the distance of f from $\mathcal{P}$ as follows:

$$\operatorname{dist}(f, \mathcal{P}) \stackrel{\text{def}}{=} \min_{g \in \mathcal{P}} \mathrm{d}(f, g) \tag{3}$$

(where $\mathrm{d}(f,g)$ *is the Hamming distance defined in Equation* (1)*). We define a testing algorithm* $\mathcal{T}$ *for monotonicity as follows: given parameter* $\epsilon \in (0,1]$*, and query access to* ***any*** f*,*

- *if* f *is monotone,* $\mathcal{T}$ *should accept (with probability* $\geq 9/10$*);*
- *if* $\operatorname{dist}(f,\mathcal{P}) > \epsilon$*,* $\mathcal{T}$ *should reject (with probability* $\geq 9/10$*).*

$\triangleright$ *As it turns out, testing monotonicity of Boolean functions can be done with* $O\left(\frac{n}{\epsilon}\right)$ *queries and runtime.*

Example 8 (Bipartiteness testing for graphs)**.** *Given an arbitrary graph* $G = ([N], E)$*, we would like to design a testing algorithm* $\mathcal{T}$ *which, given oracle access to the adjacency matrix of* G*,*

- *if* G *is bipartite, accepts (with probability* $\geq 9/10$*);*
- *if* $\operatorname{dist}(G, Bip) > \epsilon$*, rejects (with probability* $\geq 9/10$*)*

where Bip is the set of all bipartite graphs over vertex set $[N]$*, and* $\operatorname{dist}(G, Bip) = \min_{G'\in Bip} \mathrm{d}(G,G')$ *with* $\mathrm{d}(G,G') \stackrel{\text{def}}{=} \frac{|E(G)\triangle E(G')|}{\binom{N}{2}}$ *the* edit distance *between* G *and* G'*.*

$\triangleright$ *Somewhat surprisingly, testing bipartiteness can be done with* $\operatorname{poly}(\frac{1}{\epsilon})$ *queries and runtime,* independent of N.

2.3 Summary

Most of the topics covered in the class will fit into this 2×3 table:

	Boolean functions	Graphs	Probability distributions
Learning	eg, DTs		eg, monotone
Testing	eg, Monotonicity	eg, Bipartiteness	

Flavor of the course algorithms **and** lower bounds; Fourier analysis over $\{-1,1\}^n$; probability; graph theory…

3 Administrative details

See webpage: `http://www.cs.columbia.edu/~rocco/Teaching/S14/`

4 Some content – Testing Sortedness

The first topic covered will be the leftmost top cell – learning Boolean functions. However, before doing so, we will get a taste of property testing with the example of *testing sortedness of a list* (an example slightly out of the above summary table, yet which captures the "spirit" of many property testing results).

Problem: Given access to a list $\bar{a} = (a_1, \dots, a_N) \in \mathbb{Z}^N$, figure out whether it is sorted – that is, if $a_1 \leq a_2 \leq \dots a_N$. More precisely, we want to distinguish sorted lists from lists which are "far from sorted".

Definition 9. *For $\epsilon \in [0,1]$, we say a list of N integers $\bar{a} = (a_1, \dots, a_N)$ is ϵ-*approximately sorted *if there exists $\bar{b} = (b_1, \dots, b_N) \in \mathbb{Z}^N$ such that*

(i) $\bar{b}$ is sorted; and

(ii) $|\{\ i \in [N] : \ a_i \neq b_i\ \}| \leq \epsilon N$

Suggestion: write this in terms of some distance $\operatorname{dist}\left(\bar{a}, \bar{b}\right)$.

Remark 1. *This definition is equivalent to saying that $\bar{a}$ is ϵ-approximately sorted if it has a $(1-\epsilon)N$-length sorted subsequence.*

Goal: Design an algorithm which queries "few" elements a_i (here a "query" means providing the value i to an oracle, and being given a_i as response) and

- if $\bar{a}$ is sorted, accepts with high probability;
- if $\bar{a}$ is not ϵ-approximately sorted, rejects with high probability.

Remark 2. *Deterministic algorithms will not work here; indeed, consider any deterministic algorithm which reads at most $N/2$ of the a_i's; it is possible to change any sorted input on the unchecked spots to make it $\Theta(1)$-far from sorted, and the algorithm cannot distinguish between the two cases.*

Theorem 10. *There exists a $O\left(\frac{\log N}{\epsilon}\right)$-query algorithm for ϵ-testing sortedness. Further, this tester is* one-sided*:*

- *if $\bar{a}$ is sorted, it accepts with probability 1;*
- *if $\bar{a}$ is not ϵ-approximately sorted, rejects with probability $\geq 2/3$.*

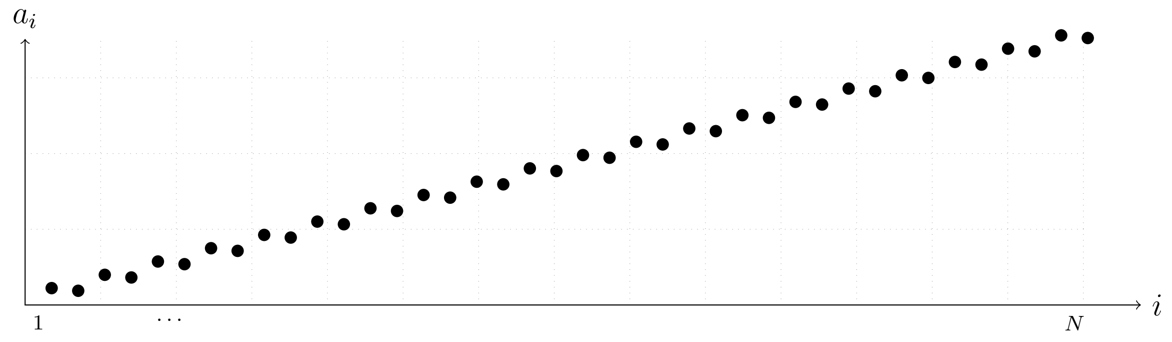
4.1 First naive attempt

Natural idea: read $\frac{\log N}{\epsilon}$ random spots, and accept iff the induced sublist is sorted.

Why does it fail? Consider the (1/2-far from sorted) list

$$\bar{a} = (11, 10, 21, 20, 31, 30, \dots, 10^{100} + 1, 10^{100}).$$

A violation will only be detected if certain consecutive elements – e.g., 21, 20 – are queried, which happens with probability $o(1)$ if we make $O\left(\frac{1}{\epsilon} \log N\right)$ queries. (Think of "ϵ" as being a small constant like 1/100.)



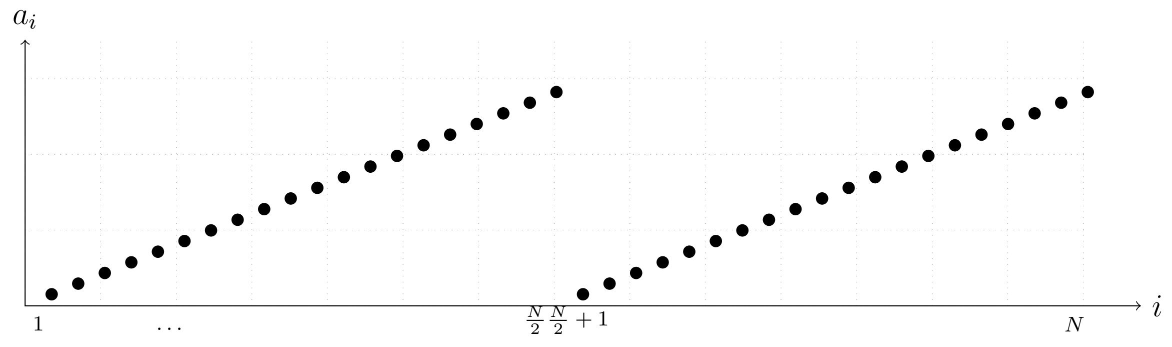
4.2 Second naive attempt

Since the previous approach failed at finding local violations, let us focus on such violations: draw pairs of consecutive elements, checking if each pair is sorted, and accept iff all pairs drawn pass the test.

Why does it fail? Consider the list

$$\bar{a} = (1, 2, 3, \dots, N/2, 1, 2, 3, \dots, N/2).$$

Once again, $\bar{a}$ is 1/2-far from sorted, yet there is only *one* pair of consecutive elements that may show a violation.



4.3 Third (and right) attempt

Some preliminary setup: first, observe that we can assume without loss of generality that all a_i's are distinct. Indeed, we can always ensure it is the case by replacing on-the-fly a_i by $b_i \overset{\text{def}}{=} Na_i + i$. It is not hard to see that the b_i's are now distinct, and moreover that $a_i \leq a_j$ iff $b_i < b_j$.

Furthermore, suppose $\bar{x}$ is a sorted list of N distinct integers. In such a list, one can use binary search to check in $\log N$ queries whether a given value x' is in $\bar{x}$.

Definition 11. *Given a (not necessarily sorted) list $\bar{a}$, we say that a_i is* well-positioned *if a binary search on a_i ends up at the i^{th} location (where it successfully finds a_i). In particular, if $\bar{a}$ is sorted, all its elements are well-positioned.*

For instance, in

$$\bar{a} = (1, 2, 100, 4, 5, 6, \ldots, 98, 99)$$

$a_3 = 100$ is not well-positioned; while $a_{75} = 75$ is.

Note that with $1 + \log N$ queries, one can query location i to get a_i and then use binary search on a_i to determine whether a_i is well-positioned.

The algorithm

```
for $\frac{10}{\epsilon}$ iterations do
    Pick $i \in [N]$ uniformly at random. Query and get $a_i$.
    Do binary search on $a_i$ ; if $a_i$ is not well-positioned, halt and return REJECT
end for
return ACCEPT
```

▷ makes $\frac{10}{\epsilon}(1 + \log N) = O\left(\frac{\log N}{\epsilon}\right)$ queries, as claimed.

- ▷ if $\bar{a}$ is sorted, all elements are well-positioned and the tester accepts with probability 1.
- ▷ It remains to prove that if $\bar{a}$ is ϵ-far from sorted, the algorithm will reject with probability at least 2/3. Equivalently, we will show the contrapositive – if $\bar{a}$ is such that $\Pr[\,\mathsf{ACCEPT}\,] \geq 1/3$, then $\bar{a}$ is ϵ-approximately sorted (i.e. has a sorted subsequence of size at least $(1-\epsilon)N$).

  Suppose that $\Pr[\,\mathsf{ACCEPT}\text{ on }\bar{a}\,] \geq 1/3$, and define $W \subseteq [N]$ to be the set of all well-positioned indices. If we had $|W| \leq (1-\epsilon)N$, then the probability that the algorithm accepts would be at most

  $$\Pr[\,\mathsf{ACCEPT}\text{ on }\bar{a}\,] \leq (1-\epsilon)^{10/\epsilon} < 0.01$$

  so it must be the case that $|W| \geq (1-\epsilon)N$. Therefore, it is sufficient to prove that the set W is sorted, that is, for all $i, j \in W$ such that $i < j$, $a_i < a_j$. Fix any two such i, j; clearly, if there were an index k s.t. (1) $i \leq k \leq j$, (2) the binary search for a_i visits k, and (3) the binary search for a_j visits k, *then* this would entail that $a_i \leq a_k \leq a_j$.
  So it is enough to argue such a k exists. To see why, note that both binary searches for a_i and a_j start at the same index $N/2$; and there must be a last common index to both searches, as they end on different indices i and j. Take k to be this last common index; as the two binary searches diverge at k, it has to be between i and j. □

References

[EKK+98] Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 259–268, New York, NY, USA, 1998. ACM.

[Ron08] Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.