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

Lecture 7: March 5, 2014

Lecturer: Rocco Servedio *Scribers: Enze Li, Yang Liu*

1 Overview

1.1 Last time

- k-juntas of prefix-coverable functions are prefix-coverable;
- gave an algorithm to learn prefix-coverable functions.

Combined, these yield a $\left(\frac{nk}{\epsilon}\right)^k$-time algorithm to learn k-juntas of halfspaces $g(h_1, \ldots, h_k)$: exponential in k, but since g is arbitrary an exponential dependence on k is unavoidable. Note that this algorithm requires membership queries.

Question: For $g = \mathsf{AND}_k$, i.e. $h_1 \wedge \cdots \wedge h_k$, is it possible to do better?

1.2 Today: Property testing for Boolean functions

- Proper learning for $\mathcal{P}$ implies property testing of $\mathcal{P}$ (generic, but quite inefficient)
- Testing linearity (over $\mathbb{GF}[2]$), i.e. $\mathcal{P} = \{\text{all parities}\}$: (optimal) $O\left(\frac{1}{\epsilon}\right)$-query 1-sided non-adaptive tester.
- Testing monotonicity ($\mathcal{P} = \{\text{all monotone functions}\}$: an efficient $O\left(\frac{n}{\epsilon}\right)$-query 1-sided non-adaptive algorithm[1].

Relevant Readings:

- Ron, 2008: *Property Testing: A Learning Theory Perspective.* [Ron08]
- Ron, 2009: *Algorithmic and Analysis Techniques in Property Testing.* [Ron09]
- Bellare, Coppersmith, Håstad, Kiwi and Sudan: *Linearity Testing in Characteristic Two.* [BCH+95].

[1]This result (from 2000) was improved to $O\left(\frac{n^{7/8}}{\epsilon^{3/2}}\right)$ by Chakrabarty and Seshadhri in 2013.

2 Proper learning implies Property Testing

Recall that a *proper learning algorithm* for a class $\mathcal{C}$ is a learning algorithm which only outputs hypotheses from $\mathcal{C}$. Hereafter, we will assume that when the algorithm fails to return such a hypothesis (e.g., after erring because the target function was not in the right class), it signals it by outputting FAIL.

Theorem 1 (Testing reduces to Proper learning)**.** *Suppose the class $\mathcal{P}$ of Boolean functions has a proper learning algorithm L that makes $m_L(n, \epsilon, \delta)$ queries and achieves $1 - \epsilon$ accuracy with probability $1 - \delta$. Then there exists a property testing algorithm T for $\mathcal{P}$ making $m_T(n, \epsilon) = m_L(n, \frac{\epsilon}{2}, \frac{1}{6}) + O\left(\frac{1}{\epsilon}\right)$ queries.*

Proof. The idea is to run L on f, getting some hypothesis h. Then, check whether h is indeed close to f on $O(1/\epsilon)$ fresh random examples, and accepts or rejects accordingly.

Algorithm 1 Generic testing algorithm

1: Run $\mathcal{L}$ on f with accuracy parameter $\epsilon/2$ and $\delta = \frac{1}{6}$. If it doesn't output $h \in \mathcal{P}$, **return** REJECT.
2: Otherwise, $h \in \mathcal{P}$: draw $O(1/\epsilon)$ uniform random examples from $\{0,1\}^n$, query f and evaluate h on them. Let $\hat{\epsilon}$ be the fraction of those examples on which f and h differ.
3: **if** $\hat{\epsilon} > 3\epsilon/4$ **then return** REJECT, otherwise **return** ACCEPT
4: **end if**

Analysis:

Case 1: $f \in \mathcal{P}$**.** With probability at least 5/6, the proper learning algorithm L will return $h \in \mathcal{P}$ such that $\text{dist}(f, h) \leq \frac{\epsilon}{2}$ – and survive Step 1. In Step 3, a (multiplicative) Chernoff bound ensures we reject with probability at most 1/6. Overall, we get that $\Pr[\, T \text{ outputs ACCEPT} \,] \geq 2/3$

Case 2: $\text{dist}(f, \mathcal{P}) > \epsilon$**.** Since L is a proper learning algorithm, every valid hypothesis h it may return will have $\text{dist}(f, h) > \epsilon$; therefore, either (a) L fails to output such a hypothesis (causing us to reject in Step 1); or (b) outputs $h \in \mathcal{P}$ with $\text{dist}(f, h) > \epsilon$, and T will reject in Step 3 with probability at least 5/6.

□

Conclusion Recall that every class $\mathcal{P}$ has a $O\big(\frac{1}{\epsilon}(\log|\mathcal{P}| + \log\frac{1}{\delta}\big)$-query proper learning algorithm (via an Occam's Razor and consistent hypothesis finder type argument). Therefore,

Good: Generic – *every* class $\mathcal{P}$ is testable with this many queries, with L;

Bad: Not optimal. It yields a $O\big(\frac{n}{\epsilon}\big)$-query tester for parities, whereas we'll see we can test this class with a customized algorithm that uses only $O\big(\frac{1}{\epsilon}\big)$ queries. For monotone Boolean functions, a bound based only on $|\mathcal{P}|$ is $2^n/(\sqrt{n}\cdot\epsilon)$, but our (non-proper) learning algorithm can be converted into a proper learning algorithm with the same query complexity, so it is possible to use the learning-based approach to get an $n^{O(\sqrt{n})}/\epsilon$-query tester. However, we will see that a customized approach for this class yields a $O\big(\frac{n}{\epsilon}\big)$-query tester.

Remark 1. *Later, [DLM+07] proved that for "many" classes $\mathcal{P}$ (those in which every function f can be approximated by a not-too-large junta), one can do a more sophisticated learning $\rightsquigarrow$ testing conversion, using junta testing ideas. For several classes such as size-s DNF formulas, size-s decision trees, and size-s Boolean circuits, this approach leads to testing algorithms with query complexity $\mathrm{poly}(s, 1/\epsilon)$ independent of n (while the straightforward "testing via proper learning" result above would give query complexities that depend on n).*

3 Linearity testing

3.1 Definitions

We consider linearity mod 2 of functions $f\colon \{0,1\}^n \to \{0,1\}$:

Definition 2. *A function f is said to be* linear *if there exists a family $(a_i)_{1\le i\le n} \in \{0,1\}^n$ such that*

$$\begin{aligned} f(x) = f(x_1,...,x_n) &= a_1x_1 + ... + a_nx_n \mod 2 \\ &= \sum_{i\in S} x_i \mod 2 \\ &= \mathrm{PAR}_S(\boldsymbol{x}) \end{aligned}$$

for $S = \{\, i : a_i = 1 \,\}$; that is, the class of all linear functions is $\mathcal{P} = \{\mathrm{PAR}_S\}_{S\subseteq[n]}$.

Athough this definition is totally fine, it is not clear how to design a testing algorithm from it (short of learning the set S). Hence, one may wonder if there is an alternate, "testing friendly" characterization of linearity – inspired by the usual characterization of linarity[2] for general vector spaces, something *local* along the lines of:

Definition 3. *A function f is said to be* linear' *if it satisfies*

$$\forall x, y \in \{0,1\}^n, \quad f(x) + f(y) = f(x+y) \tag{1}$$

where $x + y$ is the bitwise addition mod 2 (that is, $(x+y)_i = x_i + y_i \mod 2$).

Fortunately, it turns out that that these two definitions are indeed equivalent:

Lemma 4. *f is linear if and only if it is linear'.*

Proof.

$\boxed{\Rightarrow}$ Let $S \subseteq [n]$ be as in Definition 2 for f, and fix any $x, y \in \{0,1\}^n$:

$$f(x) + f(y) = \sum_{i \in S} x_i + \sum_{i \in S} y_i = \sum_{i \in S} (x_i + y_i) = f(x+y)$$

$\boxed{\Leftarrow}$ Suppose f is linear'.

- For any $x \in \{0,1\}^n$, $f(x + 0^n) = f(x) + f(0^n)$, which implies $f(0^n) = 0$;
- Define $e_i \stackrel{\text{def}}{=} (0, \ldots, 1, \ldots, 0)$, the string with only the i^{th} coordinate set to 1. For any $x \in \{0,1\}^n$ and $i \in [n]$, a simple distinction of cases show that $f(x_i e_i) = x_i f(e_i)$ (as x_i is either 0 or 1).

Setting $S \stackrel{\text{def}}{=} \{\, i \in [n] : \; f(e_i) = 1 \,\}$, we have

$$f(x) = f\Big(\sum_{i \in [n]} x_i e_i\Big) = \sum_{i \in [n]} x_i f(e_i) = \sum_{i \in S} x_i f(e_i) = \sum_{i \in S} x_i$$

proving f is linear.

□

[2]Note that the only scalars being 0 and 1, the axiom $f(\lambda x) = \lambda f(x)$ is vacuous here.

3.2 BLR Test

Observe that the second definition looks intuitively easy to test: a natural way to check linearity' is to pick x, y at random, and check whether Eq. (1) holds. This 3-query procedure, first analyzed by Blum, Luby and Rubinfeld [BLR90], is known as the *BLR linearity test*.

Algorithm 2 The BLR linearity test

Pick x, y independently and uniformly from $\{0,1\}^n$, set $z \stackrel{\text{def}}{=} x + y$ (bitwise).
Query $f(x)$, $f(y)$ and $f(z)$.
if $f(x) + f(y) = f(z)$ **then return** ACCEPT
else return REJECT
end if

The full linearity tester merely repeats the BLR tester $O(1/\epsilon)$ times, and accepts if and only if all tests passed:

Algorithm 3 (Full) linearity tester

Repeat the BLR linearity test $\frac{3}{\epsilon}$ times.
Accept if all tests passed, reject otherwise.

Clearly, this is a $9/\epsilon$-query non-adaptive tester; it remains to prove that it behaves as required. In order to show correctness, we must prove a *robust* version of Eq. (1): if f is far from linear, it has to violate $f(x) + f(y) = f(x + y)$ for many pairs (x, y).

As a first and easy observation, note that if f is indeed linear, then the test will never find a violation, hence accepting with probability 1 (i.e, this is even a one-sided tester). As for the (hard) case where f is ϵ-far from linear, it is enough to prove the following theorem on the behavior of the BLR test:

Theorem 5. *If f is such that* $\mathrm{dist}(f, \mathcal{P}) > \epsilon$, *then* $\Pr[$ *BLR outputs* YES *on* $f\,] < 1 - \epsilon$.

(indeed, as $(1-\epsilon)^{3/\epsilon} < \frac{1}{e^3} < \frac{1}{10}$, the full tester will then accept (and err) with probability at most $1/10$).

Proof. For convenience of use with Fourier analysis, we switch back to viewing f as a $\{-1,1\}^n \to \{-1,1\}$ function, and a parity PAR_S as $\chi_S(x) = \prod_{i \in S} x_i$. In this setting, the BLR test can be rephrased as checking whether $f(z) = f(x)f(y)$, for random $x, y \in \{-1,1\}^n$ and $z = x \odot y$ (that is, $z_i = x_i y_i$ for all i).

The proof will hinge upon the following lemma, which relates the probability of seccess of the BLR test to the Fourier expansion of f:

Lemma 6. *Fix any* $f\colon \{-1,1\}^n \to \{-1,1\}$. *Then*

$$\Pr[\,\textit{BLR outputs } \mathsf{Yes} \textit{ on } f\,] = \frac{1}{2} + \frac{1}{2}\sum_{S\subseteq[n]} \hat{f}(S)^3.$$

Proof. Define the indicator variable $\mathbf{1}_{\mathrm{BLR}}$ for the event "the BLR test outputs Yes on f". Rewriting "cleverly" what it means ($f(z)f(x)f(y) = 1$), we get that

$$\mathbf{1}_{\mathrm{BLR}} = \frac{1}{2} + \frac{1}{2}f(x)f(y)f(z).$$

Therefore,

$$\Pr[\,\text{BLR outputs } \mathsf{Yes}\,] = \mathbb{E}[\mathbf{1}_{\mathrm{BLR}}] = \mathbb{E}\left[\frac{1}{2} + \frac{1}{2}f(x)f(y)f(z)\right] = \frac{1}{2} + \frac{1}{2}\mathbb{E}[f(x)f(y)f(z)]$$

and, to argue the lemma, it is sufficient to show that $\mathbb{E}[(f(x)f(y)f(z))] = \sum_{S\subseteq[n]} \hat{f}(S)^3$. Expanding the Fourier expansions,

$$\begin{aligned}
\mathbb{E}[f(x)f(y)f(z)] &= \mathbb{E}\left[\sum_{S\subseteq[n]} \hat{f}(S)\chi_S(x) \sum_{T\subseteq[n]} \hat{f}(T)\chi_T(y) \sum_{U\subseteq[n]} \hat{f}(U)\chi_U(z)\right] \\
&= \sum_{S,T,U\subseteq[n]} \hat{f}(S)\hat{f}(T)\hat{f}(U)\mathbb{E}[\chi_S(x)\chi_T(y)\chi_U(x\odot y)]
\end{aligned}$$

where the expctation is over the independent choice of x and y. This last term can be itself rewritten as

$$\begin{aligned}
\mathbb{E}[\chi_S(x)\chi_T(y)\chi_U(x\odot y)] &= \mathbb{E}\left[\left(\prod_{i\in S} x_i\right)\left(\prod_{i\in T} y_i\right)\left(\prod_{i\in U} x_i y_i\right)\right] \\
&= \mathbb{E}\left[\left(\prod_{i\in S\cup U} x_i\right)\left(\prod_{i\in T\cup U} y_i\right)\right] \\
&= \mathbb{E}[\chi_{S\cup U}(x)]\,\mathbb{E}[\chi_{T\cup U}(y)] \qquad \text{(by independence of } x,y) \\
&= \begin{cases} 1, & \text{if } S=T=U \\ 0, & \text{otherwise.} \end{cases} \qquad \text{(by orthonormality of the } \chi_S\text{'s)}
\end{aligned}$$

Substituting this back into the triple-sum expression of $\mathbb{E}[f(x)f(y)f(z)]$, we get $\mathbb{E}[f(x)f(y)f(z)] = \sum_S \hat{f}(S)^3$ as desired. □

With this in hand, we are ready to prove Theorem 5 (by contrapositive). Suppose that $\Pr[\,\text{BLR outputs Yes on } f\,] \geq 1-\epsilon$: we will show this implies $\text{dist}(f,\mathcal{P}) \leq \epsilon$. Lemma 6 gives us

$$1-\epsilon \leq \Pr[\,\text{BLR outputs Yes on } f\,] = \frac{1}{2} + \frac{1}{2}\sum \hat{f}(S)^3$$

so that

$$1-2\epsilon \leq \sum \hat{f}(S)^3 \leq \max_{S\subseteq[n]} \hat{f}(S) \cdot \underbrace{\sum_{S\subseteq[n]} \hat{f}(S)^2}_{=1 \text{ (Boolean function)}}$$

Thus, there exists $S^* \subseteq [n]$ with $\hat{f}(S^*) \geq 1-2\epsilon$. Since $\hat{f}(S^*) = \mathbb{E}[f(x)\chi_{S^*}] = 2\Pr[\,f(x)=\chi_{S^*}\,]-1$, we get that $\Pr[\,f(x)=\chi_{S^*}(x)\,] \geq 1-\epsilon$, which implies $\text{dist}(f,\mathcal{P}) \leq \epsilon$. □

3.3 Generalizations

The original paper [BCH+95] and subsequent work broadly generalize this $O(1/\epsilon)$-linearity test to many other settings:

- $f\colon G \to H$, for Abelian groups G, H;
- for the class of degree-d polynomials $\mathcal{P}_d = \{\text{all degree-}d \text{ polynomials over } \mathbb{GF}_2\}$ (e.g. $f(x) = x_1x_2x_3 + x_1x_4 + x_1x_3x_5 \mod 2$): it is known that $\mathcal{P}_d$ is $\left(2^{\Theta(d)}/\epsilon\right)$-testable.

4 Monotonicity testing

We will cover the following results:

Upper Bound: We will describe and analyze the [GGL+00] $O\left(\frac{n}{\epsilon}\right)$-query (non-adaptive, 1-sided) tester. Recently, [CS13] gave a $O\left(\frac{n^{7/8}}{\epsilon^{3/2}}\right)$ tester, breaking the longstanding "barrier" of n (we won't prove this result in class).

Lower Bound For non-adaptive 1-sided testers, we will present an $\Omega(\sqrt{n})$ lower bound from [FLN+02]. For non-adaptive, 2-sided testers, the best known lower bound was for a long time $\Omega(\log n)$ from [FLN+02]. A few weeks ago, Servedio–Tan brought it up to $\Omega\left(n^{1/5}\right)$, giving the first polynomial lower bound for this problem.

Let $\mathcal{M}$ denote the class of all n variable monotone Boolean functions. Once again, for the sake of testing we will resort to a handier, equivalent definition of monotonicity:

Lemma 7. *A function $f\colon \{0,1\}^n \to \{0,1\}$ is monotone if and only if for all $x \in \{0,1\}^n$, $f(x^{i\leftarrow 0}) \leq f(x^{i\leftarrow 1})$.*

Here and subsequently the notation "$x^{i\leftarrow b}$" indicates the string in $\{0,1\}^n$ obtained from $x \in \{0,1\}^n$ by setting the i-th coordinate to b.

The algorithm will use as a subroutine the procedure EdgeTester, which picks an edge of the hypercube uniformly at random and checks whether the value of f at its two endpoints violate monotonicity:

Algorithm 4 The procedure EdgeTester

Draw independently $x \sim \mathcal{U}_{\{0,1\}^n}$, $i \sim [n]$.
Query $f(x^{i\leftarrow 0})$,$f(x^{i\leftarrow 1})$.
return REJECT if $f(x^{i\leftarrow 0}) > f(x^{i\leftarrow 1})$, ACCEPT otherwise.

Given this "edge tester", the overall testing algorithm is very simple:

Algorithm 5 The monotonicity tester MonTester.

Invoke EdgeTester $O\left(\frac{n}{\epsilon}\right)$ times.
return REJECT if any of the calls returned REJECT, ACCEPT otherwise.

Albeit very simple and "natural", the edge tester is not that straightforward to analyze.

Theorem 8. MonTester *is a $O\left(\frac{n}{\epsilon}\right)$-query, 1-sided non-adaptive tester for $\mathcal{M}$.*

Proof. The 1-sided and non-adaptive properties are easily seen to hold; the heart of the proof is, as before, in the correctness for f ϵ-far from monotone. This point will follow from the correctness of the edge tester:

Theorem 9. *Suppose* $\mathrm{dist}(f, \mathcal{M}) \geq \epsilon$. *Then,*

$$\Pr[\,\textsc{EdgeTester} \textit{ outputs } \textsc{REJECT}\,] \geq \frac{\epsilon}{n}.$$

Proof of Theorem 9.
Notation Let $E \stackrel{\text{def}}{=} \{\ (x,y)\ :\ (x,y) \text{ is an edge in } \{0,1\}^n \text{ with } x \prec y\ \}$: so that we have $|E| = n2^{n-1}$.

The set of *violating edges* $V(f) \subseteq E$ is defined as $V(f) \stackrel{\text{def}}{=} \{\ (x,y) \in E :\ f(x) = 1, f(y) = 0\ \}$. Viewing $V(f)$ as disjoint union, we write

$$V(f) = V_1(f) \cup V_2(f) \cup \cdots \cup V_n(f)$$

where $V_i(f) \subseteq V(f)$ is the set of *coordinate-i violating edges.*

Let $\eta(f) \stackrel{\text{def}}{=} \frac{|V(f)|}{|E|} = \frac{|V(f)|}{n2^{n-1}} = \Pr[\,\textsc{EdgeTester}$ outputs $\mathsf{REJECT}\,]$. Our goal is to prove that $\eta(f) \geq \frac{\text{dist}(f,\mathcal{M})}{n}$.

High-level idea: fixing some f, the proof will go by building a $g \in \mathcal{M}$ such that $\text{dist}(f,g) \leq n\eta(f)$. To do so, we will "monotonize" f, coordinate by coordinate, by sorting the violating edges for a given coordinate with a *shift operator.*

Definition 10 (Shift Operator)**.** *Fix $i \in [n]$. The* shift operator S_i *acts on functions $h\colon \{0,1\}^n \to \{0,1\}$, by sorting $h(x^{i\leftarrow 0}), h(x^{i\leftarrow 1})$: $S_i h$ is a function from $\{0,1\}^n$ to $\{0,1\}$ defined by*

$$S_i h(x^{i\leftarrow 0}) = \min\left(h(x^{i\leftarrow 0}), h(x^{i\leftarrow 1})\right)$$
$$S_i h(x^{i\leftarrow 1}) = \max\left(h(x^{i\leftarrow 0}), h(x^{i\leftarrow 1})\right)$$

(Rest of the proof during next lecture.)

□

□

References

[BCH+95] Mihir Bellare, Don Coppersmith, Johan Håstad, Marcos A. Kiwi, and Madhu Sudan. Linearity testing in characteristic two. In *FOCS*, pages 432–441, 1995.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 73–83, New York, NY, USA, 1990. ACM.

[CS13] Deeparnab Chakrabarty and C. Seshadhri. A o(n) monotonicity tester for boolean functions over the hypercube. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 411–418, New York, NY, USA, 2013. ACM.

[DLM+07] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *FOCS*, pages 549–558, 2007.

[FLN+02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *STOC*, pages 474–483, 2002.

[GGL+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

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

[Ron09] Dana Ron. Algorithmic and analysis techniques in property testing. volume 5, pages 73–205, 2009.