CS 395T Computational Complexity of Machine Learning

Lecture 18: March 28, 2005 Lecturer: Adam Klivans

Scribe: Kevin Liu

18.1 Fourier Analysis Unit Wrap-up

18.1.1 k-Juntas

Definition 1 *A k-JUNTA is any boolean function that depends on $k \ll n$ input variables.*

(Aside: a junta is a group of people who are dictators.)

For example, $f(x_1, x_2, \ldots, x_n) = x_1 x_2 + x_3 + x_4$ is a 4-junta for $n \gg 4$.

We wish to learn k-juntas in time $g(k) \cdot \mathsf{poly}(n, \frac{1}{\epsilon})$, with,

- No queries
- Uniform distribution

First notice that we can learn k-juntas in time $O(n^k 2^{2^k})$. (This brute force solution requires guessing which k variables the unknown functions depend on, then enumerating over all such functions of k variables, and finally taking enough examples.)

This question is easier than learning decision trees or DNFs. Notice that a k-junta is computed by a decision tree of size 2^k or DNF of size 2^k (since any boolean function can be written as a DNF with 2^n terms). (Aside: an algorithm for learning DNFs yields an algorithm for learning k-juntas.)

The best algorithm runs in time $n^{\frac{2k}{3}}$. For more details, see Mossel, O'Donnell, and Servedio's 2003 paper "Learning Juntas".

18.1.2 Converse to Fourier Concentration Analysis

Given a Fourier spectrum, we wish to give a DNF, decision tree, intersection of halfspaces, etc. that satisfies that Fourier spectrum. We will look at juntas to analyze this question.

Definition 2 *A boolean function f is a (γ, k)-junta if there exists a k-junta g such that f and g differ on at most $\gamma \cdot 2^n$ inputs.*

The following result is one of the deepest theorems known currently in discrete Fourier Analysis,

Theorem 1 ***Bourgain's Theorem***
For a boolean function $f : \{-1,+1\}^n \to \{-1,+1\}$, let $\delta = \mathsf{NS}_\epsilon(f)$.

If $\delta \leq \epsilon^{\frac{1}{2}+\Omega(1)}$, then f is a $\left(O(\delta), O(\delta^{-3})16^{\delta^{-4}}\right)$-junta.

The statement of this theorem is taken from Ryan O'Donnell's 2003 thesis, "Computational applications of noise sensitivity".

As the NS gets smaller and smaller, f gets closer to a junta.

Fourier analysis is used in computational complexity, NP-hard approximation analysis, coding theory, testing theory, etc. in addition to computational learning theory. For a more detailed discussion, see Ryan O'Donnell's thesis.

18.2 Query Learning

This marks the end of the Fourier analysis unit of the course and the start of the final unit in the course, where we discuss exact learning with queries. (We have black box access to an unknown concept, and we want to exactly learn that concept. In the next lecture, we will also look at learning with counterexamples, where we submit an answer to a teacher, and the teacher either accepts the answer as correct or provides a counterexample.)

The first problem we examine is polynomial interpolation.

Given an unknown polynomial $p(x_1, x_2, \ldots, x_n)$ and a field F, a learner has "black box" access to $p(x_1, x_2, \ldots, x_n)$ and must output the monomials and coefficients of p.

18.2.1 Query Learning Univariate Polynomials

In the univariate case, we wish to learn a univariate polynomial $p(x)$ of degree d.

Any distinct $d+1$ points uniquely determine p, so we will pick any $d+1$ points and interpolate them: using "black box" access to p, we obtain $d+1$ points $(b_i, p(b_i))$ for $1 \leq i \leq d+1$. Writing the unknown polynomial as $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d$, and then evaluating on these $d+1$ points, we obtain the following linear system of equations:

$$\begin{array}{rcl}
a_0 + a_1 b_1 + a_2 b_1^2 + \ldots + a_d b_1^d & = & p(b_1) \\
a_0 + a_1 b_2 + a_2 b_2^2 + \ldots + a_d b_2^d & = & p(b_2) \\
\vdots & \vdots & \vdots \\
a_0 + a_1 b_{d+1} + a_2 b_{d+1}^2 + \ldots + a_d b_{d+1}^d & = & p(b_{d+1})
\end{array}$$

Writing this more compactly,

$$\begin{pmatrix} 1 & b_1 & b_1^2 & \dots & b_1^d \\ 1 & b_2 & b_2^2 & \dots & b_2^d \\ & \vdots & & \ddots & \vdots \\ 1 & b_{d+1} & b_{d+1}^2 & \dots & b_{d+1}^d \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} p(b_1) \\ p(b_2) \\ \vdots \\ p(b_{d+1}) \end{pmatrix}$$

Taking this linear system as $V\vec{a} = \vec{p}$, we note that V is the Vandermonde matrix. It can be shown that V is invertible, and thus there exists a unique solution to this linear system to determine p.

18.2.2 Query Learning Multivariate Polynomials

Recall that a multivariate polynomial $p(x_1, x_2, \dots, x_n)$ with n variables is said to have degree d when each of the monomials in p has degree at most d.

We will proceed by reduction to the univariate case.

18.2.3 An Inefficient First Reduction

We can try making the following substitution for a first pass at a reduction,

$$x_i = y^{d^i} \qquad \text{where } d \text{ is the total degree}$$

For this substitution, every nonzero monomial of p is mapped to a unique nonzero monomial in y. (A good trick to reverse the substitution is to make a substitution zy^{d^i} for some constant z and watch which new monomial's coefficient is scaled by z.)

For example, take an example polynomial $p(x) = 3x_1x_2x_3 + 4x_1^2x_2$, which has $d = 3$. After substitution, this becomes $p(y) = 3y^3y^9y^{27} + 4y^6y^9 = 3y^{39} + 4y^{15}$. (As an example to reverse the substitution, to figure out that y^{39} corresponds to $x_1x_2x_3$, after the original substitution try making the same substitution except that $x_i = 2y^{d^i}$ and watch $3x_1x_2x_3$ become $6y^{39}$.)

Now we can interpolate $p(y)$ using $O(d^{n+1})$ queries.

Unfortunately, $O(d^{n+1})$ queries is way too large.

(The alternate substitution $x_i = y^i$ doesn't work. For instance, it will map both $x_1^2x_2$ and x_4 to the same univariate polynomial y^4.)

18.2.4 An Efficient Second Reduction

Consider the following alternate reduction. Let m be the number of monomials of p, and let d be its degree. Let q be a prime such that $q = O(m^2d)$. Pick a random $\alpha \in \mathbb{Z}_q$. Perform the following substitution,

$$x_i = y^{\alpha^i \bmod q}$$

Notice that the degree of $p(y)$ is at most $d \cdot q = \mathsf{poly}(m, d)$. (Aside: Either we can view m as part of the input, or we can double and check.)

For example, after substitution $x_1 x_2$ becomes $y^{(\alpha \bmod q + 2\alpha^2 \bmod q) \bmod q}$.

To analyze this reduction, consider what happens to a monomial in p w.r.t. the substitution.

Let $M_j = c \cdot \vec{x}^{\vec{d}^{(j)}}$ be a monomial in p, where $\vec{d}^{(j)} = (d_1, d_2, \ldots, d_n)$, i.e. d_1 is the power of x_1, d_2 is the power of x_2, etc.

After our substitution $x_i = y^{(\alpha^i \bmod q)}$, the monomial M_j will correspond to $y^{\langle \vec{d}^{(j)} \vec{a}^{(k)} \bmod q \rangle}$, for $\vec{a}^{(k)} = (1, k, k^2, k^3, \ldots, k^{q-1})$ and for a random $k \in \mathbb{Z}_q$. (Notice that there are q choices for α, corresponding to the vectors $\vec{a}^{(1)}, \vec{a}^{(2)}, \ldots, \vec{a}^{(q)}$.)

Let a monomial M after substitution be written as $M(x_i = y^{\alpha^i}) = y^{(\sum_{i=1}^{n} b_i \alpha^i \bmod q)}$, where $M = x_1^{b_1} x_2^{b_2} \ldots x_n^{b_n}$.

For a fixed monomial M, notice the following trivial upper bound on the probability of "colliding" with another monomial,

$$Pr_{\alpha \in \mathbb{Z}_q} \left[\exists m' \text{ s.t. } M(x_i = y^{\alpha^i}) = M'(x_i = y_{\alpha^i} \right] \leq \frac{mn}{q}$$

For monomial M associate a polynomial,

$$P_M(z) = \sum_{i=1}^{n} b_i z^i (\bmod q)$$

For monomial M' associate a polynomial,

$$P_{M'}(z) = \sum_{i=1}^{n} b_i' z^i (\bmod q)$$

What is the probability that P_M and $P_{M'}$ agree for a random $\alpha \in \mathbb{Z}_q$? If P_M and $P_{M'}$ agree on a point in $\mathbb{Z}_q$, then $P_M - P_{M'} = 0$ at that point. Since q is prime, there are at most n points in $\mathbb{Z}_q$ such that $P_M = P_{M'}$. Thus, for one monomial the probability in question is at most $\frac{n}{q}$.

What is the probability of any two monomials "colliding"? The following bound is similar to the first probability bound in this subsection except that there are at most m choices now for the second monomial.

$$\Pr_{\alpha \in \mathbb{Z}_q} \left[\exists M_1 \text{ and } M_2 \text{ such that } M_1(x = y^{\alpha^i}) = M_2(x = y^{\alpha^i}) \right] \leq \frac{m^2 n}{q}$$

For instance, if we choose $q \geq 100m^2n$, this implies our substitution works with probability at least $\frac{99}{100} \geq 99\%$.

Thus, WHP our reduction $x_i = y^{(\alpha^i \bmod q)}$ works and $p(x_1, x_2, \ldots, x_n) \rightarrow p(y)$ where $\mathsf{deg}(y) = O(m^2n)$. We can evaluate $p(y)$ on $O(m^2n)$ points to determine all of $p(y)$'s coefficients. (Note that given one of $O(m^2n)$ different values for y, the substitution $x_i = y^{\alpha^i \bmod q}$ directly gives the inputs to $p(\vec{x})$.)

The same coefficient trick must be applied to reverse the substitution. For example, if the substitution $x_i = y^{\alpha^i \bmod q}$ yields $21y^3 + 103y^4 + \ldots$ and the alternate substitution $x_i = 2y^{\alpha^i \bmod q}$ yields $42y^3 + 103y^4 + \ldots$, then this means y_3 came from a monomial with one x_i in it. Repeating this process for each of $1 \leq i \leq n$ allows us to determine the power of x_i corresponding to each monomial in y. (Over the real numbers, using n different primes allows us to solve this problem in one shot.)

This algorithm only works if the field is sufficiently large.

Finally, we can make this algorithm deterministic by making deterministic choices for α. Specifically, since $\alpha = O(m^2n)$ which is small, we can just enumerate over all $\alpha \in \{1, 2, \ldots, m^2n\}$.

The total runtime is $\mathsf{poly}(m, n, d)$. (Again, either m is provided as an input, or we can guess and double on m.)

18.3 Appendix A - The Vandermonde Matrix is Invertible

This can be shown by analysis on the determinant of the Vandermonde matrix V. There are several ways to do this; for example, the discussion at www-math.cudenver.edu/ rrosterm/vand/vand.html gives the determinant exactly to be $\prod_{1 \leq i < j \leq d}(b_j - b_i)$. (Noting that this expression is nonzero when for all $j \neq i$ the variables b_j and b_i are different then implies V is invertible.) Also, the invertibility is also implied by the fact that any distinct $d+1$ points uniquely determines a polynomial of degree d, which can be separately verified.

Alternately, a nice proof is due to Dexter Kozen's Algorithms course webpage,

Lemma 1 *The Vandermonde matrix V is invertible.*

Proof: We will show this by directly constructing the inverse V^{-1}.

Consider the Lagrange polynomials, for $1 \leq j \leq n$,

$$\ell_j(x) = \frac{(x - b_1)(x - b_2)\ldots(x - b_{j-1})(x - b_{j+1})\ldots(x - b_n)}{(b_j - b_1)(b_j - b_2)\ldots(b_j - b_{j-1})(b_j - b_{j+1})\ldots(b_j - b_n)}$$

Construct V^{-1} directly as the n by n matrix where the element $V(i, j)$ is the coefficient of x^i in $\ell_j(x)$.

To verify that this is the inverse, notice that $\ell_j(b_i)$ is 1 if $i = j$, and 0 otherwise. Multiplying V by the jth column of V^{-1} consisting of the coefficients of $\ell_j(x)$ yields a column vector with 1 in the jth position and 0 elsewhere, implying that indeed $VV^{-1} = I$.

■

18.4 Appendix B - References

Ryan William O'Donnell. Computational applications of noise sensitivity. Ph.D. dissertation, MIT, 2003.

E. Mossel, R. O'Donnell, R. Servedio. Learning juntas AKA Learning functions of k relevant variables, STOC '03. To appear in Journal of Computer and System Sciences.

Dexter Kozen, Design and Analysis of Algorithms. http://www.cs.cornell.edu/Courses/cs681/2004fa/.

Robert Rostermundt. The Vandermonde Matrix. http://www-math.cudenver.edu/ rrosterm/vand/vand.html.