CS 395T Computational Complexity of Machine Learning

Lecture 19: March 31, 2005 Lecturer: Adam Klivans

Scribe: HyeRyun Kim

19.1 Polynomial Interpolation

19.1.1 The probability that two monomials map to the same power of y

we talked about interpolating polynomial $P(x_1, x_2, \cdots, x_n)$ last time. The main idea is to make correct substitution to set $x_i = y^{\alpha^i \bmod q}$ for sufficiently large q. n denotes the number of variables in P, m denotes the number of monomials of P and d is the total degree of P. We let q be a prime. For example, $x_1 {x_2}^2$ maps to $y^{(\alpha \bmod q + 2\alpha^2 \bmod q) \bmod q}$.

Lemma: Given monomial $M(x_i = y^{\alpha^i}) = y^{\sum_{i=1}^n b_i \alpha^i \bmod q}$ where $M = {x_1}^{b_1} {x_2}^{b_2} \cdots {x_n}^{b_n}$.

$$\mathbf{Pr}_{\alpha \in \mathbb{Z}_q}\left[\exists M' \text{ such that } M(x_i = y^{\alpha^i}) = M'(x_i = y^{\alpha^i})\right] \leq \frac{mn}{q}$$

It means that for any fixed monomial M in original polynomial the probability that there are some other monomials map to the same power of y is at most $\frac{mn}{q}$ choosing α randomly.

Proof: Given monomial M associates a polynomial $P_M(Z) = \sum_{i=1}^n b_i Z^i \bmod q$. Monomial M' associates a polynomial $P_{M'}(Z) = \sum_{i=1}^n {b_i}' Z^i \bmod q$. Two monomials are distinct. If $P_M(Z)$ and $P_{M'}(Z)$ agree on a point in $\mathbb{Z}_q$, then $P_M(Z) - P_{M'}(Z)$ on that point equals 0. Since two distinct polynomials have degree n, $P_M(Z)$ and $P_{M'}(Z)$ can agree on at most n points. We are choosing α in $\mathbb{Z}_q$ randomly. Therefore the probability that for fixed monomial M there is one monomial M' which maps to the same value is $\frac{n}{q}$. As we have at most m monomials , there are at most m choices for M'. Therefore the probability is at most $\frac{mn}{q}$. ■

Therefore,

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

Because there are m more choices for initial fixed monomial M in lemma. If we choose $q \geq 100m^2 n$, it implies that our substitution works with probability greater than 99%.

19.1.2 The coefficients of multivariate polynomial

With high probability our reduction $x_i = y^{\alpha^i \bmod q}$ works. $P(x_1, x_2, \cdots, x_n)$ maps to $P(y)$ where degree of $y = O(m^2 n)$. Assuming we can evaluate $P(y)$, evaluate $P(y)$ on $O(m^2 n)$ points to determine all of $P(y)$'s coefficients.

How do the coefficients of $P(y)$ associate with the coefficients of $P(x_1, x_2, \cdots, x_n)$? First, substitute $x_i = y^{\alpha^i \bmod q}$. For example, we will get $21y^3 + 103y^4 + \cdots$. Then, substitute $x_i = 2y^{\alpha^i \bmod q}$. we will get $42y^3 + 103y^4 + \cdots$ in this example. It means y^3 came from a nomomial with x_i in the polynomial of example. It implies that we can get power of x_i corresponding to each monomial in y. Therefore it needs to do this for each $1 \leq i \leq n$ to determine power of x^i in y. If we are working over reals, we can use n different primes to solve this problem in 1-shot.

One thing we notice is that when we do this interpolation to get the coefficients of $P(y)$, since we know that $P(y)$ has $O(m^2n)$, we have to make sure that we are about to evaluate the polynomial over the field with at least m^2n elements in it. This algorithm is actually only going to work if the field is sufficiently large.

Let's analyze the run time of this algorithm. It takes time polynomial in (m,n,d) picking a random $\alpha \in \mathbb{Z}_q$ where $q \leq O(m^2n)$. Once we want to derandomize the algorithm, since α is small, we enumerate over all at $1, 2, \cdots m^2n$. This enumeration results in a deterministic algorithm with similar running time.

19.2 Learning Finite Automata

DFA has input $x_1x_2 \cdots x_t$ for each $x_i \in \{0, 1\}$, moves into appropriate states in DFA according to input and outputs 1 if it ends up in accept state, otherwise 0.

Learning finite automata means that learner gives input strings and gets the answer whether each input string is accepted by finite automata or not. Then learner comes up with the finite automata whose output is excatly same with that of learned finite automata. Furthermore it should learn in polynomial time.

19.2.1 Model of learning

The learner gets to ask membership queries and equivalence queries. A membership query is to ask that following string is accepted by the finite automata. A equivalence query is that learner has some candidate hypothesis and asks this hypothesis equivalent to correct automata. If the answer is yes, then learning is done. If the answer is no, then the oracle must provide counterexample t such that $h(t) \neq \mathit{Automata}(t)$. We called this model exact model with membership and equivalence queries.

Definition: Automata are efficiently learnable in this model if the learner requires at most polynomial(size of Automata) number of queries and runs in time poly(size of Automata).

It turns out that automata of size t is learnable in this model in time poly(t) due to Dana Angluin "Learning regular sets from queries and counterexamples" (1987).

19.2.2 Application 1

We can learn $\log n$-term DNF exactly polynomial-time using exact model with membership and equivalence queries. $n^{\log n}$ is best known for mistake bounded model when learning $\log n$-term DNF. How do we encode a t-term DNF which has a finite automata with 2^t states? Start out in a state corresponding to every term in t-term DNF. Every time I read a bit, say $x_1 = 0$, since it makes some terms are not satisfiable it wipes out some of my terms. Then, move to the corresponding state which has that all terms can still satisfy after reading a bit. Repeat this process. Keep track of which terms are correctly satisfiable. When you read input in the end, if you're in a state that has term left in DNF, this state is accept state.

References

[1] Adam R. Klivans and Daniel A. Spielman Randomness Efficient Identity Testing of Multivariate Polynomials