| | |
|---|---|
| **COMS 6998-2 Advanced Topics in Computational Learning Theory** | **Spring 2005** |

Lecture 6 : February 24, 2005

*Lecturer: Rocco A. Servedio* — *Scribe: Mathew Davies*

# 1 Today's Lecture

In this lecture we cover the following:

- the conclusion of the exposition of PTF lower bounds, including some open questions on PTF bounds, and
- the introduction to Uniform Distribution Learning, including some helpful facts about PAC learning, Chernoff bounds, and an algorithm for learning conjunctions.

# 2 Conclusion of PTF lower bounds

Recall the *discriminator lemma* presented in the last lecture – if $f$ is a 'nice' LTF of weight $W$ over $\{-1,1\}^n$, then for any distribution $\mathcal{D}$ there is some $x_i$ such that $|E_{x\in\mathcal{D}}[f(x)x_i]| \geq \frac{1}{W}$.

**Outline of Proof** The proof of the $\Omega(\frac{\log n}{\log\log n})$ PTF degree bound for $\mathrm{MAJ}(x_1,...,x_n) \wedge \mathrm{MAJ}(y_1,...,y_n)$ makes use of the discriminator lemma.

- WLOG, suppose $n$ is odd.
- Suppose $p(x_1,...,x_n,y_1,...,y_n)$ is a degree-$d$ PTF for $\mathrm{MAJ}(\bar{x}) \wedge \mathrm{MAJ}(\bar{y})$.
- By symmetrizing, we can get a degree-$d$ polynomial $q(u,v)$ such that for all odd integers $u,v$ with $1 \leq |u|,|v| \leq n$, we have

$$q(u,v) > 0 \text{ if } u > 0 \text{ and } v > 0$$

$$q(u,v) < 0 \text{ if } u < 0 \text{ or } v < 0$$

So

$$q(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i) = \sum_{\pi}\sum_{\sigma} p(\pi(x), \sigma(y))$$

where $\pi, \sigma$ are permutations of the input vector.

- Now let $\mathbb{Z}_n^{odd} \equiv \{-n, -n+2, ..., -1, 1, 3, ..., n-2, n\}$, and consider the sign of $q(u,v)$ across the four quadrants of $\mathbb{Z}_n^{odd} \times \mathbb{Z}_n^{odd}$. Only in the first quadrant will $q$ be positive.

  The proof constructs a probability distribution $\mathcal{D}'$ over $\mathbb{Z}_n^{odd} \times \mathbb{Z}_n^{odd}$ such that for all pairs of nonnegative integers $(a,b)$ with $a+b \leq \frac{1}{9}\frac{\log n}{\log\log n}$ [1], we have

  $$E_{(u,v)\in\mathcal{D}'}[u^a v^b \text{sign}(q(u,v))] = 0;$$

  that is, under $\mathcal{D}'$, for each low-degree monomial there is no correlation between the monomial and the function that outputs $+1$ in the first quadrant and $-1$ in the other quadrants.

- The discriminator lemma gives us that the degree of $q$ is at least $\frac{1}{9}\frac{\log n}{\log\log n}$:

  Suppose for the purpose of contradiction that $\deg(q) \leq \frac{1}{9}\frac{\log n}{\log\log n}$. Clearly

  $$E_{(u,v)\in\mathcal{D}'}[q(u,v)\text{sign}(q(u,v))] > 0$$

  (because $q(u,v)\text{sign}(q(u,v))$ is always positive). However, expanding the polynomial as $q(u,v) = \sum_{a,b\in\mathbb{Z}'} \alpha_{a,b} u^a v^b$ gives that

  $$\begin{aligned} E_{(u,v)\in\mathcal{D}'}[\sum_{a,b\in\mathbb{Z}'} \alpha_{a,b} u^a v^b \text{sign}(q(u,v))] &= \sum_{a,b\in\mathbb{Z}'} \alpha_{a,b} E_{(u,v)\in\mathcal{D}'}[u^a v^b \text{sign}(q(u,v))] \\ &= 0 \end{aligned}$$

  (The RHS of this equation is zero because the quantity $E_{(u,v)\in\mathcal{D}'}[u^a v^b \text{sign}(q(u,v))]$ is zero by the prior construction of $\mathcal{D}'$). Hence, it must be that $\deg(q) > \frac{1}{9}\frac{\log n}{\log\log n}$.

[1] The constant $\frac{1}{9}$ in this and later expressions with the double log may not be exact.

## 2.1 open questions concerning PTF bounds

1. What is the PTF degree of $\mathrm{MAJ}(x_1, ..., x_n) \wedge \mathrm{MAJ}(y_1, ..., y_n)$?

   Note that the degree $d$ has been nailed down to within

   $$\Omega(\frac{\log n}{\log\log n}) \leq d \leq \Omega(\log n)$$

   but can this be improved?

2. Come up with 2 LTF's $h_1$, $h_2$ and prove a $\omega(\frac{\log n}{\log\log n})$ PTF bound on $h_1 \wedge h_2$.

   Note that if the weight of each halfspace is at most $W$, there is an $O(\log W)$ upper bound. If $h_1$ and $h_2$ are decision lists, the weight is $2^{\Omega(n)}$, but $h_1 \wedge h_2$ has a degree-2 PTF. (Imagine the second decision list 'hanging' from each positive result of the first decision list; this makes a rank-2 decision tree, and it was shown earlier that any rank-$r$ decision tree has a degree-$r$ PTF corresponding to it.)

3. Give an $o(n)$-degree PTF upper bound on the AND of any two halfspaces.

   Note that the $\Omega(\frac{\log n}{\log\log n})$ lower bound rules out a BRS-style approach to this problem:

   If $O(\log W)$ could be made $O(\sqrt{\log W})$, then since any LTF over $\{-1,1\}^n$ has weight $2^{O(n\log n)}$, for any two LTFs you would get

   $$O(\sqrt{\log W}) \approx O(\sqrt{\log 2^{O(n\log n)}}) \approx O(\sqrt{n\log n})$$

   Given the lower bound result just proved, however, a BRS approach would start with at the very best $O(\frac{\log W}{\log\log W})$, which for $W = 2^{n\log n}$ reduces to $O(n)$. So an $o(n)$ bound is not attainable in this way (and even showing $O(n)$ via this approach would require an improvement of the BRS $O(\log W)$ bound to $O(\frac{\log W}{\log\log W})$).

4. Give *any* $2^{o(n)}$-time algorithm for learning an AND of two halfspaces.

This is the end of PTF degree bounds. Now we turn to a new unit.

# 3 Uniform Distribution Learning

Uniform distribution learning (UD learning) is a restricted variant of PAC learning.

Note the following readings for this lecture: (i) KV chapter 1 §4.1 and appendix 9 for more on the PAC (probably approximately correct) learning model and related tools; (ii) Mansour '94 and Verbeurgt '88.

The basic model of UD learning differs from online learning in several ways.

- The input examples are chosen randomly from a distribution, rather than being chosen by some adversary.
- The goal of learning is to approximate the target concept to within some arbitrary accuracy, rather than to find it exactly.
- The distribution from which examples are taken is the *uniform distribution* $\mathcal{U}$ over $\{-1,1\}^n$, which puts weight $\frac{1}{2^n}$ on every $x \in \{-1,1\}^n$.

## 3.1 definitions and notation

**Definition 1** *Given* $f : \{-1,1\}^n \to \{-1,1\}$, *a* uniform example oracle *for* $f$, *written EX(f), is a 'black box' that when called draws an example* $x$ *from the uniform distribution* $\mathcal{U}$ *over* $\{-1,1\}^n$ *and outputs the ordered pair* $(x, f(x))$.

Note that when the oracle is called multiple times in succession, each successive draw is independent.

**Definition 2** *Given functions* $f, h : \{-1,1\}^n \to \{-1,1\}$, *the* error *of* $h$ *on* $f$ *is*

$$\mathit{error}(h,f) \equiv Pr_{x \in \mathcal{U}}[h(x) \neq f(x)] = \frac{|\{x \in \{-1,1\}^n : h(x) \neq f(x)\}|}{2^n}$$

**Definition 3** *Given any concept class* $C$ *over* $\{-1,1\}^n$, *a* uniform distribution learning algorithm *for* $C$ *is an algorithm* $A$ *with the following functionality.*

- *$A$ is given as inputs the values* $0 < \epsilon, \delta < 1$ *and access to a uniform example oracle EX(c), where* $c \in C$ *is unknown to* $A$.
- *$A$ outputs a* hypothesis $h$ *such that with probability at least* $1 - \delta$, $\mathit{error}(h,f) \leq \epsilon$.

**Notation.**

| | | |
|---|---|---|
| $m$ | – | the *sample complexity* of the algorithm (number of samples required to get the desired output hypothesis, as a function of parameters of the algorithm) |
| $\epsilon$ | – | the *accuracy* of the output hypothesis |
| $\delta$ | – | the *confidence* in the accuracy of the output hypothesis |

Note the following important considerations, which are true for PAC learning generally.

- Runtime: an *efficient* PAC algorithm should run in $\text{poly}(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$. Clearly, the runtime order is at least $\Omega(m)$.
- Evaluatability: a PAC algorithm may utilize any form of hypothesis representation that is *efficiently evaluatable*, i.e. if a PAC algorithm $A$ runs in $p(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ for some polynomial $p$, then evaluating $h$ on any $x$ should take at most $p(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ time.
- Generality: uniform distribution learning is a special case of PAC learning, so results that are valid for general PAC learning are also valid for UD learning. (N.B.: The converse of this statement is not generally true.)

## 3.2 useful facts about PAC learning

The following facts, which are true for general PAC learning, will be useful for getting results on UD learning.

1. **Fact 1** *A PAC algorithm's dependence on $\delta$ can always be made $O(\log \frac{1}{\delta})$.*

   How is this possible? Suppose we want to get arbitrarily small confidence and accuracy of $\epsilon'$ and $\delta'$ respectively. Then we can use an algorithm $A$ with input parameters $\delta = \frac{1}{4}, \epsilon = \frac{\epsilon'}{2}$ and run it $T = O(\log \frac{1}{\delta'})$ times. Because each hypothesis of the series $h_1, ..., h_T$ was independently generated, the probability that none of them is $\frac{\epsilon'}{2}$-accurate will be at most $\frac{\delta'}{2}$ (with a suitable choice of $T$).

   Now run each of the $h_1, ..., h_T$ on $M$ fresh examples to estimate the error rate, and pick as $h$ the $h_i$ with the lowest error rate. We can show that, for a suitable value of $M$, $\Pr[\text{error}(h_i) > \epsilon'] \leq \frac{\delta'}{2}$.

   Hence, we can more or less disregard $\delta$ in finding good algorithms.

2. **Fact 2** *Ignoring runtime, it's easy to learn any reasonable (finite) concept class with a small number of examples.*

   We can motivate this fact with the following algorithm.

   **Algorithm 1** *(Elimination-like PAC learning)*

   - *Draw $M$ examples from an example oracle.*
   - *Search all $h \in C$ until you find one that gets all $M$ examples right.*

   Why does this work? Well, suppose that $c \in C$ is the target concept, and let's call $h \in C$ *bad* if $\Pr[h(x) \neq c(x)] > \epsilon$. Then for any fixed bad $h$,

   $$\Pr[h \text{ gets } M \text{ examples right}] \leq (1-\epsilon)^M$$

   There are at most $|C|$ bad hypotheses, so

   $$\Pr[\text{any bad } h \text{ is output}] \leq |C|(1-\epsilon)^M$$

   To satisfy the PAC criteria, require the right hand side above to be at most $\delta$. The value $M = \frac{1}{\epsilon}(\ln C + \ln \frac{1}{\delta})$ satisfies this requirement.

   For example, if $C$ is $\{$all $s$-term DNF formulas over $x_1, ..., x_n\}$, then $|C| = 3^{ns}$, giving a sample complexity of $O(ns)$. This means that we can learn $C$ to arbitrary accuracy with only a polynomial-size sample of labeled examples. This leads us to the following observation about the character of PAC learning.

   **Observation 1** *What we are really interested in here is not information theory, but rather computational complexity. Learning in this domain is hard not because there isn't enough information, but because making use of that information is often not computationally very tractable.*

3. **Fact 3** *Any online learning algorithm can be converted into a PAC learning algorithm.*

   We can express this precisely as follows.

   **Theorem 1** *If $A$ is an online algorithm that learns concept class $C$ with mistake bound $M$ in time $T$, then there is a PAC algorithm that learns $C$ using $O(\frac{m}{\epsilon}\log\frac{1}{\delta})$ examples and runs in poly$(\frac{1}{\epsilon}, T, \log\frac{1}{\delta})$.*

   **Sketch of Proof** Run $A$ on a stream of $N = \frac{m}{\epsilon}\log\frac{1}{\delta}$ examples from $\mathrm{EX}(c, \mathcal{D})$ to produce a set of hypotheses. Draw $N$ examples again and test each hypothesis on them; output the hypothesis with the lowest error rate. It can be shown that this hypothesis satisfies the PAC accuracy and confidence criteria. ■

## 3.3 Chernoff bounds

Chernoff bounds are frequently invoked in PAC learning. Why are they useful? Well, given a hypothesis $h$, we'd like to know the value of $\text{error}(f, h)$ by estimation: draw $m$ examples from $\text{EX}(f, \mathcal{D})$, and count how many of them are right; this gives the fraction $\hat{\epsilon}$ of the $m$ examples that are wrong. But how confident can we be that the estimate $\hat{\epsilon}$ is close to the true error $\epsilon$?

This is exactly what Chernoff bounds tell us. They come in two varieties, multiplicative and additive.

**Definition 4** *Let $X_1, ..., X_m$ be independent random boolean variables, each with $Pr[X_i = 1] = p$. Let $X = \sum_{i=1}^{M} X_i$. Then for $0 \leq \gamma \leq 1$,*

$$Pr[X \leq (1-\gamma)mp] \leq e^{-\frac{1}{2}\gamma^2 mp}$$

$$Pr[X \geq (1-\gamma)mp] \leq e^{-\frac{1}{3}\gamma^2 mp}$$

These are the *multiplicative Chernoff bounds*[2]; they are useful when $p$ is very small.

**Definition 5** *Let $X_1, ..., X_m$ be independent random boolean variables, each with $Pr[X_i = 1] = p$. Let $X = \sum_{i=1}^{M} X_i$, and let $\hat{p} = \frac{X}{m}$. Then*

$$Pr[p - \hat{p} \geq \epsilon] \leq e^{-2m\epsilon^2}$$

$$Pr[\hat{p} - p \geq \epsilon] \leq e^{-2m\epsilon^2}$$

These are the *additive Chernoff bounds*; they are useful when $p$ is not too small.

## 3.4 learning conjunctions

Now we turn to the problem of learning conjunctions.

Let $C = \{\text{all conjunctions over } x_1, ..., x_n\}$. For example, $c(x) = \bar{x_1} \wedge x_5 \wedge x_6$. We'd like to learn $c$ to an accuracy of $\epsilon$. (Note that if $c = x_1 \wedge x_2 \wedge ... \wedge x_n$, we would need at least $2^n$ examples to know that $c$ is not identically zero). We use the following algorithm.

[2] See Motwani/Raghavan for derivations of these bounds.

**Algorithm 2** *(UD learning conjunctions)*

- *Given $\epsilon > 0$, draw poly($\frac{1}{\epsilon}, \log\frac{1}{\delta}$) many examples to estimate $Pr_{x\in\mathcal{U}}[c(x) = 1]$ to within an additive factor of $\pm\frac{\epsilon}{4}$. If the estimate is $< \frac{\epsilon}{2}$, output "$h \equiv 0$" (with error $< \frac{3\epsilon}{4}$).*
- *Otherwise, we can assume that $Pr_{x\in\mathcal{U}}[c(x) = 1] \geq \frac{\epsilon}{4}$, so we know that every $O(\frac{1}{\epsilon})$ examples (on average) gives at least one positive example.*
- *We say that $x_j$ is* bogus *if it does not appear in the target conjunction $c$. Given $k$ independent positive examples, if $x_j$ is bogus, then*

  $$Pr[x_j \textit{ has the same value in all } k \textit{ examples}] = \frac{1}{2^{k-1}}$$

  *So*

  $$Pr[\textit{any bogus variable has the same value in all } k \textit{ examples}] \leq \frac{n}{2^{k-1}}$$

  *We require $\frac{n}{2^{k-1}} < \delta$, which implies that*

  $$k = O(\log\frac{n}{\delta})$$

(To be continued...)