Analysis of Boolean Functions **(CMU 18-859S, Spring 2007)**

Lecture 8: Learning under the uniform distribution

Feb. 8, 2005

Lecturer: Ryan O'Donnell *Scribe: Moritz Hardt*

1 The Learning Model

A learning problem is identified with a *(concept) class* $\mathcal{C}$ of functions $f : \{-1,1\}^n \to \{-1,1\}$.

A learning algorithm A is a (usually randomized) algorithm with 2 inputs

$\epsilon > 0$ accuracy parameter

$\delta > 0$ confidence parameter

and access to some target function f:

Either: **Random Examples**	Or: **(Membership) Queries**
A can ask for an example and gets a pair $(\boldsymbol{x}, f(\boldsymbol{x}))$ where $\boldsymbol{x}$ is drawn from the uniform distribution.	A can ask for f's value on any string x it wants.

Finally, A outputs a *hypothesis* $h : \{-1,1\}^n \to \{-1,1\}$ in the form of a circuit.

- We say, A *learns* $\mathcal{C}$ if for all ϵ, δ and for all $f \in \mathcal{C}$ when A is run, the output h satisfies that it is ϵ-close to f with probability at least $1 - \delta$.
- A is efficient if it runs in time $\mathrm{poly}(n, 1/\epsilon, 1/\delta)$.

Remark 1.1 *The PAC ("Probably Approximately Correct") Learning model is due to Valiant:* A theory of the learnable. *C.ACM 1984.*

We will only consider the case of PAC-learning under the uniform distribution. This manifests itself in two places:

- accuracy of the hypothesis: $\mathbf{Pr}_{\boldsymbol{x} \text{ unif. on } \{-1,1\}^n}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})]$
- distribution of random examples: $(\boldsymbol{x}, f(\boldsymbol{x}))$ where $\boldsymbol{x}$ is drawn uniformly at random.

General PAC-learning insists that one algorithm works simultaneously for all distributions. In fact, the machine learning community interested in real-world applications finds the uniform setting questionable: "They'll punch you in the nose if you try to tell them about algorithms in this framework." –Ryan. But, in a more general framework, no one can really prove anything.

We think of uniform distribution learning as a part of complexity theory and cryptography rather than as part of real-world Machine Learning.

Remark 1.2 *The random examples model is the traditional access model. The membership queries model can be appropriate though, e.g., in attacking crypto systems.*

2 Run time

The trivial run time is $O(n2^n)$. Even with just random examples, you will see the output of the target function on every input with high probability so that you can learn with $\epsilon = 0$.

What we want instead is a run time polynomial in n and $1/\epsilon$. We don't care much about n^2 vs. n^3 or n^4.

As it turns out, the parameter δ is *never interesting*. Inevitably, any algorithm runs $T(n, 1/\epsilon)$ many steps. Some steps interact with the examples (usually estimating something about f) and can "fail", but the failure probability can be decreased to δ' at the cost of $O(\log 1/\delta')$ repetitions. So, set $\delta' = \delta/T(n, 1/\epsilon)$. Then, except with probability δ, no steps fail (union bound). The total running time overhead is multiplicative $\log(T(n, 1/\epsilon)) + \log(1/\delta)$. Also, algorithms can test their own hypotheses. So, any running time dependence on δ (like $2^{2^{1/\delta}}$) can be reduced to $\log(1/\delta)$. We will henceforth ignore δ!

Although we want $\mathrm{poly}(n, 1/\epsilon)$ run time, a lot of the classes we're interested in are "pretty hard". So, sometimes we will settle for:

- Poly-time assuming ϵ is "constant", e.g., $n^{O(1/\epsilon)}$.
- Quasipolynomial time: $(n/\epsilon)^{\log(n/\epsilon)}$.

This is, for instance, relevant for cryptography where even quasipolynomial time adversaries should not be successful.

3 Interesting Classes

Mainly, we fix a *representation* format (for boolean functions $f : \{-1, 1\}^n \to \{-1, 1\}$) and then set the concept class $\mathcal{C}$ to be the class of functions with $\mathrm{poly}(n)$ size in that format. Let us briefly recall the definition of two particularly interesting representations.

Definition 3.1 *A* DNF formula *is a disjunction* $\phi = T_1 \vee T_2 \vee \ldots T_s$ *where the* terms T *are a conjunction of* literals*, e.g.,* $x_1 \wedge \bar{x_3} \wedge x_2$.

- *The* size *of a DNF is the number of terms.*
- *The* width *is the maximum number of literals in any term.*

Definition 3.2 Decision trees *are rooted trees where every inner vertex is labeled with a variable name* x_i *and has two outgoing edges labeled with* -1 *or* 1*. Every leaf is labeled with an output value* -1 *or* 1*. An assignment to the variables defines a path in the decision tree from the root to an output value in the obvious way. We assume no variable appears twice on any path.*

- *The* depth *of a decision tree is the maximum number of variables on any path in the tree.*
- *Its* size *is the number of leaves in the tree.*

Fact 3.3 • *If f has a decision tree of depth d, then it is of size at most 2^d.*

- *Decision trees of size s are s-juntas.*
- *If f has a decision tree of size s, then it also has a DNF of size s, i.e., "DNF-size$(f) \leq$ DT-size(f)".*
- *An r-junta has a depth-r decision tree and a width-r, size 2^r DNF.*

We get the following interesting learning problems:

- Learning functions with $\mathrm{poly}(n)$ size decision trees.
- Learning functions with $\mathrm{poly}(n)$ size DNF formulas.

Remark 3.4 *Especially natural are universal representation formats: All functions can be represented with* some *size. Then, we often talk about learning* all functions *in time* $\mathrm{poly}(n, 1/\epsilon, s)$, *where s is the size of the smallest such representation for the function. That is, the more complicated the function, the more time you are allowed. In fact, we don't even need to know s; try $s = 1, 2, 4, 8, \ldots$ and "check".*

4 Best known algorithms

Class	Random Examples		Membership Queries	
poly-size depth-d circuits	$n^{O(\log^{d-1}(n/\epsilon))}$	(1)	same	
poly-size DNF	$n^{O(\log(n/\epsilon))}$	(2)	$\mathrm{poly}(n/\epsilon)$	(3)
poly-size DTs	$n^{O(\log(n/\epsilon))}$		$\mathrm{poly}(n/\epsilon)$	(4)
$\log(n)$-juntas	$n^{.704\log(n)+O(1)}$	(5)	$\mathrm{poly}(n)$	

1. [LMN'93]: We will prove this modulo Håstad's "Switching Lemma".

   [Kharitonov'93]: Assuming factoring "takes exponential time", this can't be done, even with queries, in time better than $n^{\log^{\Omega(d)} n}$.

2. This follows from the above. [Verbeurgt'90]

3. Celebrated result of Jeff Jackson. We will prove an earlier result of Mansour: $n^{O(\log\log n)}$ (assuming ϵ constant).

4. We will prove this before the Mansour result. A nice application of Goldreich-Levin and Fourier analysis.

5. People would kill to do DTs in polynomial time from random examples, but even doing $\log(n)$-juntas in polynomial time is conjectured to be hard. Avrim Blum will give you *\$1000* if you can do it. [BFKL] built a cryptosystem assuming it. The marginal $.704$ improves trivial results from crypto.

5 Learning via spectral concentration

As we will see, we can get learning algorithms for several classes of boolean functions by analyzing the *concentration* of the fourier spectrum of those functions.

Definition 5.1 *Given a collection $\mathcal{S}$ of subsets of $[n]$, we say f has ϵ-*concentration *on $\mathcal{S}$, if*

$$\sum_{S \notin \mathcal{S}} \hat{f}(S)^2 \leq \epsilon.$$

Proposition 5.2 *In this case, $g : \{-1,1\}^n \to \mathbb{R}$ defined by $g = \sum_{S \in \mathcal{S}} \hat{f}(S)\chi_S$ satisfies*

$$\|f - g\|_2^2 \leq \epsilon.$$

Proof:

$$\|f - g\|_2^2 = \mathop{\mathbf{E}}_{\boldsymbol{x}}[((f-g)(\boldsymbol{x}))^2] = \sum_{S \subseteq [n]} \widehat{f-g}(S)^2 = \sum_{S \notin \mathcal{S}} \hat{f}(S)^2 \leq \epsilon$$

□

Proposition 5.3 *Let $f : \{-1,1\}^n \to \{-1,1\}$ and $g : \{-1,1\}^n \to \{-1,1\}$ satisfy $\|f-g\|_2^2 \leq \epsilon$. Then, for $h : \{-1,1\}^n \to \{-1,1\}$ defined by $h(x) = \mathrm{sgn}(g(x))$, f and h are ϵ-close.*

Proof: Suppose that for more than an ϵ fraction of the x's, we have $h(x) \neq f(x)$. For such x, $(f(x) - g(x))^2 \geq 1$. But then,

$$\mathop{\mathbf{E}}_{\boldsymbol{x}}[((f-g)(\boldsymbol{x}))^2] > \epsilon.$$

□

Theorem 5.4 *Suppose an algorithm A can "somehow" find a collection $\mathcal{S}$ on which f is $\epsilon/2$-concentrated. Then, in further time $\mathrm{poly}(|\mathcal{S}|, 1/\epsilon, n)\log(1/\delta)$, using only random examples, A can output a hypothesis h such that with probability at least $1 - \delta$ the functions h and f are ϵ-close.*

Proof: The algorithm proceeds as follows:

1. For each $S \in \mathcal{S}$, it estimates $\hat{f}(S)$ to within $\pm\sqrt{\epsilon}/(4\sqrt{|\mathcal{S}|})$ in time $\mathrm{poly}(|\mathcal{S}|, 1/\epsilon, n)$.

   Call the estimate $\widetilde{\hat{f}(S)}$.

2. The algorithm constructs $\widetilde{g}(x) = \sum_{S \in \mathcal{S}} \widetilde{\hat{f}(S)} x_S$ and outputs $h = \mathrm{sgn}(\widetilde{g})$.

For the purpose of analysis, let $g = \sum_{S \in \mathcal{S}} \hat{f}(S)\chi_S$. We know $\|f - g\|_2^2 \leq \epsilon/2$. And,

$$\begin{aligned}
\|g - \widetilde{g}\| &= \sum_S \widehat{g - \widetilde{g}}(S)^2 \\
&= \sum_{S \in \mathcal{S}} (\hat{g}(S) - \hat{\widetilde{g}}(S))^2 \\
&= \sum_{S \in \mathcal{S}} \left(\hat{f}(S) - \widetilde{\hat{f}(S)} \right)^2 \\
&\leq |\mathcal{S}| \frac{\epsilon}{16|\mathcal{S}|} = \frac{\epsilon}{16}.
\end{aligned}$$

By the triangle inequality,

$$\|f - \widetilde{g}\| \leq \|f - g\|_2 + \|g - \widetilde{g}\|_2 \leq \sqrt{\epsilon/2} + \sqrt{\epsilon/16} \leq \sqrt{\epsilon}.$$

Therefore, $\|f - \widetilde{g}\|_2^2 \leq \epsilon$, that is, h is ϵ-close to f. $\square$

Corollary 5.5 1. *If you can prove for all $f \in \mathcal{C}$, f is $\epsilon/2$-concentrated on $\mathcal{S} = \{S \subseteq [n] : |S| \leq d(\epsilon/2)\}$, then $\mathcal{C}$ can be learned in time $\mathrm{poly}(|S|, 1/\epsilon) \leq n^{O(d)} \cdot \mathrm{poly}(1/eps)$ using random examples only.*

2. *If you can prove for all $f \in \mathcal{C}$, f is $\epsilon/4$-concentrated on some collection of cardinality at most M, then $\mathcal{C}$ can be learned in time $\mathrm{poly}(M, 1/\epsilon)$ using membership queries.*

Proof: (2) Let $\mathcal{T}$ be the collection of size M of sets S with the largest $\hat{f}^2$. Then, f is ϵ-concentrated on $\mathcal{T}$. Using the Goldreich-Levin algorithm, find a collection $\mathcal{L}$ containing all S such that $\hat{f}(S)^2 \geq \epsilon/4M$ in time $\mathrm{poly}(M, 1/\epsilon)$.

We are done, if can prove f is $\epsilon/2$-concentrated on $\mathcal{L}$.

$$\sum_{S \notin \mathcal{L}} \hat{f}(S)^2 = \sum_{S \notin \mathcal{L}, \mathcal{S} \in \mathcal{T}} \hat{f}(S)^2 + \sum_{S \notin \mathcal{L}, \mathcal{S} \notin \mathcal{T}} \hat{f}(S)^2 \leq M \cdot \frac{\epsilon}{4M} + \frac{\epsilon}{4} = \frac{\epsilon}{2}$$

$\square$

Exercise: If $\|f - g\|_2^2 \leq \epsilon$ and g is ϵ-concentrated on $\mathcal{S}$, then f is $O(\epsilon)$-concentrated on $\mathcal{S}$.

Proposition 5.6 *Suppose $f : \{-1,1\}^n \to \{-1,1\}$ with $\mathbb{I}(f) \leq d$. Then,*

$$\sum_{|S| > d/\epsilon} \hat{f}(S)^2 \leq \epsilon.$$

Proof: Remember, $\mathbb{I}(f) = \sum_S |S| \hat{f}(S)^2$. So, if $\sum_{|S| > d/\epsilon} \hat{f}(S)^2 > \epsilon$, then $\mathbb{I}(f) > d$. $\square$

Corollary 5.7 *The class $\mathcal{C} = \{f : \mathbb{I}(f) \leq d\}$ is learnable in time $n^{O(d/\epsilon)}$ using random examples only.*

Proposition 5.8 *If f has a decision tree of depth at most d, then $\mathbb{I}(f) \leq d$.*

Proof: For any given x, at most d coordinates are influential. □

Corollary 5.9 *Decision trees of depth $\log(n)$ are learnable in time $n^{O(\log(n)/\epsilon)}$ using random examples only.*

Homework: If f has a DNF of width d, then $\mathbb{I}(f) \leq 2d$.

Corollary 5.10 *DNFs of with $\log(n)$ are learnable in time $n^{O(\log n/\epsilon)}$ for random examples.*

Proposition 5.11 *If f has a decision tree of size s, then f is ϵ-close to some g with DT-depth at most $\log(s/\epsilon)$.*

Proof: Given the DT for f, truncate paths longer than $\log(s/\epsilon)$. The truncated tree computes some function g. But, for any path of length $\log(s/\epsilon)$ the probability that a random x follows this path is bounded by $2^{-\log(s/\epsilon)} = \epsilon/s$. Therefore, by union bound,

$$\mathbf{Pr}_{\boldsymbol{x}}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})] \leq s \cdot \frac{\epsilon}{s} = \epsilon.$$

□

Corollary 5.12 *Decision trees of size s are learnable from random examples in time*

$$n^{O(\log(s/\epsilon)/\epsilon)} \approx n^{O(\log n)}.$$