CIS 625 Problem Set 1

Lingqi Zhang (lingqiz@sas.upenn.edu)

1. • Suppose C_1 is PAC learnable by H_1 with algorithm L_1: we can assume given m positive example, L_1 will output $h_1 \in H_1$ s.t. with probability $\geq 1-\delta_1$, $\varepsilon(h_1) \leq \varepsilon_1$, with running time $\text{poly}_1(m, n)$

   • Similarly, assume given m positive example, L_2 will output $h_2 \in H_2$ s.t. with prob $\geq 1-\delta_2$, $\varepsilon(h_2) \leq \varepsilon_2$, with running time $\text{poly}_2(m, n)$

   • We construct algorithm L_3 that takes m positive example as input. since $C_1(x) \wedge C_2(x) = 1$, $C_1(x) = C_2(x) = 1$. we call L_1 and L_2 with those m positive example, and return $h_1 \wedge h_2$ as the output.

• The running time for our algorithm is $\text{poly}_1(m, n) + \text{poly}_2(m, n)$ which is still polynomial. Assume L_1 and L_2 make error independently with each other, then with probability at least $1-(\delta_1+\delta_2)$, we have $\varepsilon(h_1 \wedge h_2) \leq \varepsilon_1 + \varepsilon_2$.

---

2. • Assume we have algorithm L_1 and L_2 similar to that of Q1, except for now L_2 requires both positive

and negative example.

- We can construct L_3 as follows: L_3 takes m positive and m' negative examples. We first run L_1 with the m positive examples (since $c_1(x)=1$ if $c_1(x) \wedge c_2(x)=1$) and get h_1. We run h_1 on the m' negative examples and obtain $m'' < m'$ examples for which $h_1(x')=1$, we can infer $c(x')=0$. We then use m positive and m'' negative examples as input to L_2, and get h_2. The final output of L_3 is $h_1 \wedge h_2$.
- Similar to Q1, assume the errors are independent, then with probability $\geq 1-(\delta_1+\delta_2)$, we have $\varepsilon(h_1 \wedge h_2) \leq \varepsilon_1 + \varepsilon_2$.

---

3. In general, $C_1 \vee C_2$ is NOT PAC learnable by $H_1 \vee H_2$.

Proof by contradiction:

- ① Suppose $C_1 \vee C_2$ is indeed PAC learnable by $H_1 \vee H_2$ in general if C_1 is PAC learnable by H_1 and C_2 is PAC learnable by H_2.
- ② We know that C = conjunction of literals over boolean $x_1, \ldots x_n$ is PAC learnable by $H=C$ from positive examples only.

- ② Denote $T_1 \in C$, $T_2 \in C$, combining ① ② we have that $T_1 \vee T_2$ is PAC learnable by $C \vee C$.
- Denote $T_3 \in C$, combining ① ② we have that $T_1 \vee T_2 \vee T_3$ is PAC learnable by $C \vee C \vee C$.
- Thus, for ① to be true, we will have 3-term DNF $(T_1 \vee T_2 \vee T_3)$ is PAC learnable by 3-term DNF, which will imply RP = NP. This is in contradiction with the widely believed assumption that RP ≠ NP. Thus, ① cannot be true in general.

---

4. Assume there is an algorithm L for PAC learning monotone 3-term DNF $h_M \in H_M$.

For training examples $(X_1 X_2 \dots X_n, Y)$ we can expand them to have the following form:

$$(X_1 X_2 \dots X_n \neg X_1 \neg X_2 \dots \neg X_n, Y)$$

We can use the expanded form of training examples as input to algorithm L, and with probability $\geq 1-\delta_0$, it outputs h_M with $\varepsilon(h_M) \leq \varepsilon_0$. The computational complexity will be $O(2n \cdot m)$.

Since h_M defined on the expanded input represents

the exact same class of function of 3-term DNF on the original input, L is thus a PAC learning algorithm also for 3-term DNF by 3-term DNF.

In general, if a finite length monotone Boolean formulae C is PAC learnable, C is PAC learnable in general.

---

5. Similar to 4, we can expand the input (X, Y) to the following form:

$X_1 \dots X_N \neg X_1 \dots \neg X_N \; X_1 \dots X_N \neg X_1 \dots \neg X_N \; X_1 \dots X_N \neg X_1 \dots \neg X_N$

(each X and its negation occur 3 times).

Then we can use the read-once PAC learning algorithm to learn 3-term DNF in general. The computational complexity will be of the form $O(6n \cdot m)$.

In general, for Boolean formulae C, if its length is finite and each variable only appears a fixed number of times, then PAC learning its read-once counterpart means C is also PAC learnable.

---

6. we construct the following algorithm L':

given a fixed number δ, ε and N:

- Run algorithm L with sample size m s.t. with probability $\geq 1 - \sqrt[N]{\delta}$, L outputs h with $\varepsilon(h) \leq \varepsilon$.
- Evaluate h, if $\varepsilon(h) \leq \varepsilon$, then stop and return h as the final output, otherwise, run L again with m new sample. Repeat for a maximum of N times.

Analysis: Since each time L outputs h with $\varepsilon(h) > \varepsilon$ with probability $\sqrt[N]{\delta}$, and each run of L is statistically independent (since we take a new set of sample). Thus, L' will fail with probability at most: $(\sqrt[N]{\delta})^N = \delta$. That is, with prob $\geq 1 - \delta$, L' outputs h with $\varepsilon(h) \leq \varepsilon$.

In terms of sample complexity, we will only need to run L with the new $\delta' = \delta^{\frac{1}{N}}$. For any value of δ, we can find N sufficiently large such that:

$$N \cdot \text{poly}(1/\delta^{\frac{1}{N}}) \leq \text{poly}[\log(1/\delta)]$$

---

1. Since C is PAC learnable by C, thus for any distribution

P, and constant δ, ε, there is a randomized polynomial algorithm that with prob $\geq 1-\delta$, returns c with $\varepsilon(c) < \varepsilon$.
To solve the consistency problem with arbitary set S of size $|S| = N$, we adopt the following:

① Choose P_0 s.t. for $x_i \in S$, $p(x_i) = \frac{1}{N}$ (and for $x_i \notin S$, $p(x_i) = 0$).

② Choose ε_0 s.t. $\varepsilon_0 < \frac{1}{N}$. According to ①, any c with $\varepsilon(c) < \frac{1}{N}$ under P_0 must be consistent with sample in S.

③ Choose δ_0 s.t. $\delta_0 \leq 0.01$.

Call the PAC learning algorithm with P_0, ε_0 and δ_0. By definition, with prob $\geq 1-\delta \geq 0.99$, the algorithm returns C_0 that is consistent with S with a polynomial running time.

---

8. For sample S of size $|S| = m$, we can divide S into many dataset of $|S_1| = m_1$, $|S_2| = m_2$, ...
we can run L on $|S_1|$ to obtain h_1
$|S_2|$ to obtain h_2

so on and so forth.

We can choose m_i carefully s.t. h_i returned by the algorithm is sublinear in m_i.

We can combine h_i's into $h = h_1 \wedge h_2 \ldots \wedge h_n$