Leslie G. Valiant
TFs: Aayush Karan and Kevin Cong

CS 2280: Computational Learning Theory
Homework 1
Due: Feb. 19, 11:59pm
Nico Fidalgo

Note For some problems you may need the following fact from probability theory called the Chernoff bound:

Theorem 1 (Chernoff bound)**.** *Let X_i, $1 \leq i \leq m$, be independent Bernoulli random variables each with probability of success μ. Then for any $\lambda \in [0, 1]$,*

$$\Pr\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| \geq \lambda\mu\right] \leq 2e^{-\lambda^2\mu m/2}$$

You can find more details and generalizations in the textbook or online.

Policy reminders You are strongly encouraged to type your solutions using LaTeX. You may discuss problems with your classmates, but not merely copy each others solutions. You must write all solutions by yourself, list your collaborators on your problem sets and also appropriately cite any resources outside of the class materials that you have used. You are not allowed to look up solutions to the problems. Please do not use LLMs or LLM-assisted tools for finding solutions to the problems.

Problem 1. (10pt) **Learning union of intervals.** Give a PAC-learning algorithm for the concept class $\mathcal{C}_2$ formed by unions of two closed intervals, that is $[a, b] \cup [c, d]$, with $a, b, c, d \in \mathbb{R}$. Extend your result to derive a PAC-learning algorithm for the concept class $\mathcal{C}_n$ formed by unions of $n \geq 1$ closed intervals, thus $[a_1, b_1] \cup \cdots \cup [a_n, b_n]$, with $a_i, b_i \in \mathbb{R}$ for $k \in [n]$. What are the time and sample complexities of your algorithm as a function of n ?

To give a PAC-learning algorithm for the union of n intervals, the goal is to learn a function that correctly classifies points as either inside or outside the target intervals while ensuring an error of at most ϵ with confidence at least $1 - \delta$.
Given a set of positive samples $S = \{x_1, x_2, \ldots, x_m\}$, we construct the hypothesis by identifying pairs of adjacent positive samples. We assume there are negative samples in between the intervals to separate them, $\{c_0, c_1, \ldots, c_n\}$. The hypothesis $h(x)$ labels x as 1 if it lies in an interval $[x_i, x_j]$ and 0 otherwise.
To control the error, we must ensure that the positive samples sufficiently cover the intervals, particularly near their boundaries. If an interval lacks positive samples near its edges (in a fraction of length $\frac{\epsilon}{2}$), it may lead to classification errors. The probability of missing samples in any of these $2n$ regions is bounded by

$$P(\text{missing any of } 2n \text{ tails}) \leq 2nP(\text{missing one tail})$$

Which, using probability bounds, reduces to

$$P(\text{miss one tail}) \leq e^{-\frac{m\epsilon}{2}}$$

At most, this probability is δ, therefore the sampling complexity is

$$\frac{2}{\epsilon}\ln\frac{2n}{\delta} \leq m$$

This means that the number of positive samples needed grows logarithmically with the number of intervals n, i.e., $O(\log n)$.
The number of negative samples needed is at most $O(n)$ since we only need to place one between each pair of adjacent intervals. The time complexity of processing all samples is therefore $O(m+n)$ which simplifies to $O(n)$ as m grows as $O(\log n)$.

Problem 2. (10pt) **PAC-learning parity functions.** *Parity functions* is analogous to conjunctions or disjunctions, but with the XOR operation $\oplus$ rather than $\wedge$ or $\vee$. To be precise, given a subset $S \subseteq \{1, 2, \ldots, n\}$, a parity function $f_S : \{0,1\}^n \to \{0,1\}$ outputs 1 iff the number of 1s in the subset of the input indexed by S is odd. For example, if $n = 4$ and $S = \{1, 3, 4\}$, the parity function $f_S = x_1 \oplus x_3 \oplus x_4$ will return $f(0010) = 1, f(1010) = 0$. The class of parity functions $\mathcal{P}$ includes all functions that can be described in this way:

$$\mathcal{P} = \{f_S : S \in 2^{\{1,\ldots,n\}}\} \qquad f_S(x) = \bigoplus_{i \in S} x_i$$

Show that $\mathcal{P}$ is PAC learnable.

Hint. Note that $x \oplus y$ is equal to $x + y \pmod 2$. Additionally, searching for an Occam algorithm might be easier than working with ϵ and δ.

Each parity function is defined by a subset $S \subseteq \{1, \ldots, n\}$. Learning the function means discovering the unknown set S using examples $(x, f_S(x))$ where x is an n-bit binary vector and $f_S(x)$ is the parity (XOR sum) of the bits in S, i.e., $f_S(x) = \bigoplus_{i \in S} x_i$. The goal is to figure out S from training data.
The XOR operation is equivalent to addition modulo 2, i.e.,

$$x \oplus y = (x + y) \mod 2$$

If we collect enough labeled examples $(x, f_S(x))$ then we can solve a system of linear equations to determine S. To learn S, we observe that each training example $(x, f_S(x))$ gives us one linear equation with n unknowns, i.e.,

$$x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_k} = f_S(x)$$

The algorithm is to collect n independent samples $(x, f_S(x))$ and use them to construct a system of n linear equations in n variables and solve for S using Gaussian elimination.
Solving for S requires at most n independent equations, thus we only need $O(n)$ samples. Gaussian elimination runs in $O(n^3)$ time, so the overall time complexity is $O(n^3)$. As Gaussian elimination is deterministic, solving for S is guaranteed to be correct, therefore satisfying the PAC-learning requirement.

Problem 3. (10pt) **"Relaxed" δ-dependence in PAC definition.** Prove that if a concept class $\mathcal{C}$ is PAC-learnable by class $\mathcal{H}$ by the usual definition (in time polynomial in $1/\delta, 1/\epsilon, n, \text{size}(c)$), then it is in fact learnable by class $\mathcal{H}$ in time polynomial in $\log(1/\delta), 1/\epsilon, n, \text{size}(c)$. (Notice the more stringent requirement on the running time's dependence on δ.)

Hint. Can we boost our confidence by running the original PAC-learning algorithm multiple times?

We start with a PAC-learning algorithm that outputs a hypothesis function h satisfying the PAC-learning requirements. Instead of relying on a single hypothesis, we run the algorithm k times and gather multiple candidate hypotheses $h_1, h_2, \ldots, h_k$. Each hypothesis is independetly trained on different data samples.
To create a more robust classifier, we can use majority voting. Given an input x, we compute $h_i(x)$ for each of the k hypotheses and the final output is determined by whichever label appears most frequently. Since each hypothesis is independent, the probability of many of them making an error at the same time follows a Chernoff bound. If h has error upper-bounded by ϵ, the probability that a majority of k hypotheses failing is at most $e^{-\frac{\epsilon k}{2}}$.
The probability of the majority of hypotheses failing is at most δ if we set $k = \frac{2}{\epsilon} \ln \frac{2}{\delta}$. The time complexity of running the algorithm k times is $O(k \cdot \text{time}(n, \epsilon, \delta))$ which is polynomial in $\log(1/\delta), 1/\epsilon, n, \text{size}(c)$. This means we can achieve the same accuracy with confidence boosted exopnentially, leading to a sample complexity that depends on $\log(1/\delta)$.

Problem 4. (10pt) **Approximate Occam algorithm.** For constants $\alpha \geq 1$, $\beta < 1$, and $\gamma < 1/2$, let a γ-approximate (α, β)-Occam algorithm be the same as an (α, β)-Occam algorithm, but instead of consistency with all the examples only consistency with at least $1 - \gamma$ fraction of examples is guaranteed. Similarly let a γ-PAC learning algorithm for a concept class $\mathcal{C}$ be the same as a PAC learning algorithm except that it produces a $(\gamma + \epsilon)$-approximate (instead of an ϵ-approximate) hypothesis.

Show that if there is a γ-approximate (α, β) Occam algorithm for $\mathcal{C}$ then there is a γ-PAC learning algorithm for C.

Problem 5. (15pt) **Learning unions of halfspaces is NP-hard.** A *linear threshold function* or a *halfspace* is a function $f(x) : \{0,1\}^n \to \{0,1\}$ such that there exists some n-dimensional real vector w and real number θ such that $f(x) = 1$ iff $w \cdot x \geq \theta$. A union of halfspaces is simply a disjunction of several halfspaces. Suppose that for some $k \geq 3$, unions of k halfspaces are efficiently PAC learnable by an algorithms that outputs unions of k halfspaces. Show that then there is a randomized algorithm R taking as input a graph G that runs in time polynomial in the size of G with the following properties:

- If G is k-colorable then with probability at least $2/3$, R outputs a k-coloring of G.
- If G is not k-colorable, then R's output may be arbitrary.

Hint. Consider adding the example $\langle 1^n, - \rangle$ to the reduction you saw in class.

Problem 6. (15pt) **Conjunction of PAC learnable classes is PAC learnable.** Show that if both class C_1 and class C_2 are PAC learnable from positive examples only (each separately), then class $C = \{c_1 \wedge c_2 \mid c_1 \in C_1 \text{ and } c_2 \in C_2\}$ is also PAC learnable from positive examples only.

Hint. Let $h = h_1 \wedge h_2$, where h_1, h_2 are obtained from feeding the positive examples of $c_1 \wedge c_2$. What do we know about the relationship between c_1 and h_1?

Problem 7. (15pt) **Probabilistic PAC is equivalent to deterministic PAC.** Let $\mathcal{C}$ be any concept class and $\mathcal{H}$ be any hypothesis class. Let h_0 and h_1 be representations of the identically 0 and identically 1 functions respectively. Show that if there is a randomized algorithm for efficiently PAC learning $\mathcal{C}$ using $\mathcal{H}$, then there is a deterministic algorithm for efficiently PAC learning $\mathcal{C}$ using $\mathcal{H} \cup \{h_0, h_1\}$.

Note: There are two sources of randomness we are talking about.
1. The inherent randomness that occurs because the example oracle EX is random.
2. The randomization that the algorithm uses. You are asked to get rid of the second randomization. More precisely if you run the learning algorithm several times using the same sequence of examples as output by the example oracle, the final hypothesis your output should be the same (and almost accurate with high probability).

This problem is complex, so we will help you organize the solution as below. Observe that a randomized algorithm can be viewed as a deterministic algorithm which takes as additional input a random string, which we may regard as a sequence of fair coin toss. The algorithm is then required to work correctly with high probability over this input string. Let the randomized algorithm be L and consider the following deterministic algorithm:

Using an oracle $EX(c, \mathcal{D})$ get M data points. Set $i = 1$.
If less than $\epsilon M/2$ examples have label 0 return h_1.
Else if less than $\epsilon M/2$ examples have label 1 return h_0.
Else

- Whenever L requests an example return (x_i, y_i) and increment i.
- Whenever L requests a random bit as follows:
  - If $y_i \neq y_{i+1}$, then return y_i and increment i twice.
  - Otherwise, increment i twice, effectively discarding these two points.

  If L terminates, return the hypothesis output by L. Else whenever $i > M$, i.e. we run out of data points, output h_0.

There are now three sources of error from this deterministic algorithm:

- Incorrectly choosing h_0 or h_1 as the hypothesis at the start;
- L terminates but returns a hypothesis with a large error;
- L fails to terminate before running out of M examples.

For simplicity, you may assume that L requires m examples and t random bits to output with probability at least $1 - \delta/3$ a hypothesis with error at most ϵ. Note that in reality m and t are not necessarily fixed, but in order for L to be a PAC-learning algorithm, they must be polynomial in all the relevant parameters. Therefore, the second source of error is taken care of.

1. (3pt) Explain the algorithm for generating a random bit above. Why does it produce a fair coin toss?
2. (5pt) Find a lower bound on M such that if we choose h_0 or h_1 as the final hypothesis at the start, then with probability at least $1 - \delta/3$, we will have error at most ϵ.
3. (7pt) Find a lower bound on M such that assuming we do not choose h_0 or h_1 as the final hypothesis at the start, with probability at least $1 - \delta/3$ we will generate at least t random bits using $M - m$ examples.