Leslie G. Valiant
TFs: Aayush Karan & Kevin Cong

CS 228: Computational Learning Theory
Homework 2
Due: March 12, 11:59PM

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 (10pts). **Efficient Evaluability is Necessary.** The definition of learning in the PAC model requires that hypotheses produced by a learning algorithms are evaluatable efficiently (that is, in polynomial time). In this problem you will show that this requirement is necessary for a meaningful notion of learning.

Suppose we consider relaxing this restriction, and let $\mathcal{H}$ be the class of all Turing machines (not necessarily polynomial time). Show that if $\mathcal{C}_n$ is the class of all Boolean circuits of size at most $p(n)$ for some fixed polynomial $p(\cdot)$, then $\mathcal{C}$ is efficiently PAC learnable using $\mathcal{H}$.

Argue that your solution shows that this relaxation trivializes the model of learning. (Hint: How could you use your solution to trivially learn other concept classes we've seen in class?)

To show that the requirement that the hypotheses produced by a learning algorithm are efficently evaluatable in polynomial time, consider the following scenario: Suppose we have a learning algorithm $\mathcal{L}$ that efficiently PAC learns the concept class $\mathcal{C}_n$ of Boolean circuits with size bounded by $p(n)$:

Algorithm $\mathcal{L}$:

- **Input:** A sample $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ where each $y_i = c(x_i)$ for some target concept $c \in \mathcal{C}_n$
- **Output:** A Turing machine T_S that implements a hypothesis h

Definition of T_S: For any input $x \in \{0,1\}^n$, T_S executes:

1. Systematically enumerate all Boolean circuits $c' \in \mathcal{C}_n$ in order of increasing circuit size
2. For each circuit c', check if $c'(x_i) = y_i$ for all $(x_i, y_i) \in S$
3. Return $c'(x)$ for the first circuit c' is found that is consistent with S

Proof $\mathcal{L}$ is an Occam Algorithm: Since the target concept $c \in \mathcal{C}_n$ is consistent with S, the algorithm will find at least one consistent circuit (possibly c itself or another circuit that agrees with c on all examples in S). Also, the description size of T_S is polynomial in n and m. Specifically, the Turing machine T_S can be described in $O(n + \log m)$ bits as it only has to encode the enumeration procedure for circuits (constant-sized code), the sample S (requires at most $O(m \cdot n)$ bits), and the consistency checking logic (constant-sized code). Thus, for some constants α and β, we have $\text{size}(T_S) \leq (n \cdot \text{size}(c))^{\alpha} \cdot m^{\beta}$. Lastly, constructing T_S takes polynomial time since we only need to encode the sample and search procedure. Therefore, $\mathcal{L}$ is an Occam algorithm that efficiently PAC learns $\mathcal{C}_n$ using $\mathcal{H}$ as the hypothesis class.

Conclusion: This approach can be generalized to learn *any* concept class that has a finite representation. For any learnable concept class $\mathcal{D}$, we can define a similar Turing machine that enumerates all concepts in $\mathcal{D}$ and returns the first one consistent with the sample. Despite such a Turing machine having a compact description making the training process efficient, evaluating it on new examples would be computationally expensive. The Turing machine could have to do an exhaustive search through an exponentially large search space of possible concepts before producing an output. Thus, without the efficiently evaluatable constraint, we could trivially learn any concept class by outputting a hypothesis that performs a brute-force search through the concept space-a strategy that doesn't reflect the true nature of learning.

Problem 2 (15pts)**. Infinite Mistake-Bounded Model.** The *Infinite Mistake-Bounded (IMB) model* is a Boolean on-line learning model where the number of attributes in the world may be infinite. It is assumed, however, that each example has a finite number of attributes assigned 1. An example is presented to the learner as a list of its positive attributes. The learning scenario is the same as in the (standard) mistake-bounded model.

Provide an on-line learning algorithm that makes at most $\mathcal{O}(k \log n)$ mistakes when learning any monotone disjunction of k literals, if every example presented to the learner has at most n positive attributes.

Note: Assume for full credit that the learner does not know the value of n in advance. For partial credit, you can use the value of n in your algorithm

COME BACK TO THIS PROBLEM

Problem 3 (15 points)**. Function Classes: Linear Thresholds, DNFs, and Decision Lists.**

a) Prove that any 1-decision list over $x_1, \ldots, x_n$ can be expressed as a linear threshold function $\mathbb{1}[w \cdot x \geq \theta]$, where $w \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, $\theta \in \mathbb{R}$.

b) Prove that the class of linear threshold functions and the class of poly(n)-term DNF are incomparable in that neither class is contained in the other.

COME BACK TO THIS PROBLEM

Problem 4 (15 points)**. Perceptrons Can Make Exponentially Many Mistakes.**

a) The *margin* δ of a set of points $X \subseteq \{0,1\}^n$ labeled by a function f is defined as follows.

$$\delta = \max_{w \in \mathbb{R}^n, \|w\|_2 = 1,\ \theta \in \mathbb{R}} \{\delta' \mid \forall x \in X,\ |w \cdot x - \theta| \geq \delta' \text{ and } (w \cdot x \geq \theta \text{ iff } f(x) = 1)\}$$

(This is the δ of the Perceptron algorithm.) Give a set of $O(n)$ examples on $\{0,1\}^n$ that are linearly separable but for which the margin is exponentially small in n.

b) Give an example of a linear threshold function on which the Perceptron algorithm can make exponentially many mistakes.

Part (a): To find examples that are linearly separable but have exponentially small margin, I'll create two classes of points in $\{0,1\}^n$. The classification rule f will be such that a point $x \in \{0,1\}^n$ has $f(x) = 1$ if and only if the first non-zero element appears at an odd-index position. The first class of points we'll call $\{a_1, a_2, \ldots, a_n\}$ where a_i has its first $i-1$ components as 0 and all remaining components as 1. Mathematically this can be described as:

$$a_1 = (1, 1, \ldots, 1) \tag{1}$$
$$a_2 = (0, 1, 1, \ldots, 1) \tag{2}$$
$$\vdots \tag{3}$$
$$a_n = (0, 0, \ldots, 0, 1) \tag{4}$$

The second class of points will be $\{b_1, b_2, \ldots, b_n\}$ where b_i has its first $i-1$ components as 0 followed by alternating 0's and 1's. Mathematically this can be described as:

$$b_1 = (1, 0, 1, 0, \ldots) \tag{5}$$
$$b_2 = (0, 1, 0, 1, \ldots) \tag{6}$$
$$\vdots \tag{7}$$
$$b_n = (0, 0, \ldots, 0, 1) \tag{8}$$

According to our classification rule, points a_i and b_i are labeled 1 when i is odd and 0 when i is even. To correctly classify these points, any weight vector w must have $w_i > 0$ when i is odd and $w_i < 0$ when i is even. And specifically for the b_i points, the magnitude of w_i must dominate the sum of all subsequent weights: $|w_i| > \sum_{j>1} |w_j|$. These constraints on w force it to decay exponentially. A valid weight vector would be $w = (2^{-1}, 2^{-2}, 2^{-3}, \ldots)$. Then, after normalizing the vector to unit length, the margin becomes $\delta = O(2^{-n})$ which is exponentially small in n.

Part (b): The Perceptron algorithm's mistake bound is inversely proportional to the square of the margin. Consider the weight vector $w = \left(1, -\frac{1}{2}, \frac{1}{4}, -\frac{1}{8}, \ldots, \frac{(-1)^{n-1}}{2^{n-1}}, 0\right)$. When normalized to unit length, this defines a linear threshold function $f(x) = 1$ if $w \cdot x \geq \theta$ and $f(x) = 0$ otherwise, where $\theta = 0$. For this function, the weights alternate signs and decrease exponentially in magnitude. The margin $\delta_w = \min_{x \in X} |w \cdot x|$ is less than 2^{-n}, and the squared norm $\|w\|_2 = \sum_i w_i^2$ converges to a constant. By the Perceptron mistake bound theorem, the number of mistakes is bounded by $\frac{\|w\|^2}{\delta_2^2}$ which is at least 2^{2n} for this weight vector. Thus, the Perceptron algorithm can make exponentially many mistakes on this linear threshold function.

Problem 5 (10pts)**. Logarithmic Mistake Bounds.**

a) Let $\mathcal{C}_n$ be a finite concept class. Give a learning algorithm for $\mathcal{C}_n$ which has mistake bound $\log_2 |\mathcal{C}_n|$. Your algorithm need not be computationally efficient.

b) Show a concept class $\mathcal{C}$ for which there exists an algorithm with a mistake bound that is asymptotically better than $\log_2 |\mathcal{C}_n|$.

Part (a): Algorithm $\mathcal{L}$:

1. Initialize $H = \mathcal{C}_n$ (all possible hypotheses)
2. For each input x:
   (a) Compute $h(x)$ for each $h \in H$
   (b) Predict the majority vote of all hypotheses in H
   (c) If the prediction is incorrect, eliminate all hypotheses in H that made the same mistake

Initially $|H| = |\mathcal{C}_n|$ but after each mistake, at least half of the hypotheses in H are eliminated (as we predict with the majority so if a mistake is made, all hypotheses that voted with the majority were incorrect). The target concept always remains in H as we never eliminate the correct hypothesis. Thus, since $|H|$ starts at $|\mathcal{C}_n|$ and is reduced by at least half after each mistake and $|H| \geq 1$ always as the target concept always remains in H, the maximum number of mistakes is bounded by $\log_2 |\mathcal{C}_n|$.

Part (b): Consider the concept class $\mathcal{C}_n$ that represents labelings of vertices on a regular polygon with a multiple of 3 number of vertices (triangle, hexagon, nonagon, etc.) where each vertex is labeled with 0 or 1 and valid concepts only assign label 1 to vertices that form a contiguous segment of exactly $2n$ connected neighboring points. Altogether, this concept class has exactly $3n$ possible concepts (one for each possible starting position of the $2n$ segment around the polygon). When using the halvign algorithm for this concept class, for any input vertex, exactly $2n$ out of the $3n$ concepts will label it as 1. So if the algorithm makes a mistake on a prediction, it must eliminate either all concepts that predicted 1 if the true label was 0 which eliminates $\frac{2n}{3n} = \frac{2}{3}$ of the concepts, or all concepts that predicted 0 if the true label was 1 which eliminates $\frac{n}{3n} = \frac{1}{3}$ of the concepts. Therefore, in the worst case 2/3 of the remaining hypotheses are eliminated with each mistake, bounding the number of mistakes by $\log_3 |\mathcal{C}_n|$. This is asymptotically better than $\log_2 |\mathcal{C}_n|$.

Problem 6 (15 points). **Robust Winnow Algorithm.** In this question we examine the error robustness of the Winnow algorithm. Let f be a monotone disjunction of k variables. Suppose that an adversary manipulates the labels of some of the examples seen by the Winnow algorithm as it runs on f: during the online learning process, for some inputs x, the Winnow algorithm is told that the label of x is $1 - f(x)$ when the correct value was actually $f(x)$. When $f(x) = 1$ but the adversary flips its reported label to 0, we call this is a *false negative example.* When $f(x) = 0$ but the algorithm is presented with the label 1, this is called a *false positive example.*

Assume that the algorithm sees s false negative examples and t false positive examples during the course of its learning. Prove that the total number of mistakes the Winnow algorithm[1] will make (on examples that are labeled correctly) is $O(k \log n + ks + t)$.

Note. You will have to modify the Winnow algorithm to get this result. In class the demotion step set all weights $w_i = 0$ if $x_i = 1$. Argue why this will not work in the above setting and show that if instead you set $w_i \leftarrow w_i/2$, you will get the required bounds.

In the standard Winnow algorithm, when a false positive occurs, the weights of active variables are set to 0 ($w_i = 0$ if $x_i = 1$). While this works well in noise-free environments, it's problematic with adversarial noise because a single false positive example would cause the algorithm to make no mistake but incorrectly demote weights, or worse, a false negative example would cause the algorithm to set the weights of relevant variables to 0. Once a relevant variable's weight is set to 0, it remains 0 forever essentially removing that variable from consideration. With adversarial noise, this could permanently eliminate relevant variables from the hypothesis, making it impossible to learn the target concept. Instead, I'll use a modified version where during demotion, weights are halved rather than zeroed.

- Initialize all weights $w_i = 1$
- Predict 1 if $\sum_{i=1}^{n} w_i x_i \geq \theta$ where θ is the threshold
- If false negative, i.e., predict 0 when true label is 1, then $w_i \to 2w_i$ for all i where $x_i = 1$
- If false positive, i.e., predict 1 when true label is 0, then $w_i \to w_i/2$ for all i where $x_i = 1$

[1] if modified appropriately; see the note at the end of the problem statement.

Mistakes on correctly labeled examples (normal learning) is one source of mistakes. In this case, each relevant variable can be promoted at most $\log n$ times since weights start at 1. As there are k relevant variables in the disjunction, this results in at most $k \log n$ mistakes from normal learning. In false negative examples, weights of relevant variables can be incorrectly halved. When this happens, it might require a promotion to become relevant again. In the worst case, all k relevant variables might need to be promoted again, so for each of the s false negative examples, there's a potential amplification of k mistakes, resulting in at most $k \cdot s$ additional mistakes. For false positives, the weights of irrelevant variables are mistakenly increased. Each such example can lead to at most one additional mistake when the algorithm encounters a correctly labeled negative example, so with t false positive examples, we get at most t additional mistakes. So altogether, we get $O(k \log n + ks + t)$ mistakes.

Problem 7 (15pts)**. VC Dimension of Linear Halfspaces.** Let $\mathcal{C}$ be the concept class of linear halfspaces in $\mathbb{R}^n$. A halfspace is specified by an inequality of the form $c(x) = \mathbb{1}[w \cdot x \geq \theta]$, where $w \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and $\theta \in \mathbb{R}$. Prove that the VC dimension of $\mathcal{C}$ is $n + 1$.

Note. You may use the following result without proof:

Theorem 1 (**Radon's theorem**)*. Let $S = \{x^{(1)}, \ldots, x^{(m)}\} \subset \mathbb{R}^n$ be a set of m points in $\mathbb{R}^n$. The* convex hull *of S is the set*

$$\{z \in \mathbb{R}^n \mid \exists \lambda_1, \ldots, \lambda_m \text{ s.t. } z = \sum_{i=1}^{m} \lambda_i x^{(i)}, \text{ with each } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1\}$$

If $m \geq n + 2$ then S must have two disjoint subsets S_1 and S_2 whose convex hulls intersect.

MY ANSWER