6.842 Randomness and Computation March 31, 2022

Homework 4

Lecturer: Ronitt Rubinfeld *Due Date: April 13, 2022*

Homework guidelines: You may work with other students, as long as (1) they have not yet solved the problem, (2) you write down the names of all other students with which you discussed the problem, and (3) you write up the solution on your own. No points will be deducted, no matter how many people you talk to, as long as you are honest. If you already knew the answer to one of the problems (call these "famous" problems), then let me know that in your solution writeup – it will not affect your score, but will help me in the future. It's ok to look up famous sums and inequalities that help you to solve the problem, but don't look up an entire solution.

The following problem is not for turning in – just for fun!

1. **(Quadratic non-residuosity)** Let Z_n^* be the group of integers that are relatively prime with n. An element $s \in Z_n^*$ is said to be a *quadratic residue* modulo n if there exists $r \in Z_n^*$ s.t. $s \equiv r^2 \bmod n$. Give a private-coin interactive proof system for the language of pairs (s, n) such that s is *not* a quadratic residue modulo n.

The following problems are for turning in:

1. **Random bipartite graphs are good vertex expanders:** A graph $G = (V, E)$ is called an (n, d, c)-vertex expander if it has n vertices, the maximum degree of a vertex is d and for every subset $W \subseteq V$ of cardinality $|W| \leq n/2$, the inequality $|N(W)| \geq c|W|$ holds, where $N(W)$ denotes the set of all vertices in $V \setminus W$ adjacent to some vertex in W. By considering a random bipartite 3-regular graph on $2n$ vertices obtained by picking 3 random permutations between the 2 color classes, one can prove that there exists $c > 0$ such that for every n there exists a $(2n, 3, c)$-vertex expander.

   For this homework, you are asked to prove that for any subset L of size at most $n/2$ of the "left vertices", (with constant probability) there are at least $(1 + \epsilon)|L|$ "right" neighbors. In other words, you should show that the construction succeeds with constant probability (you can use $1/2$ for instance), for some sufficiently small constant $\epsilon = \Theta(1)$.

   Note that it is fine to admit multi-edges in the construction.

2. **Testing Dictator functions:** *Dictator functions*, also called *projection functions*, are the functions mapping $\{+1, -1\}^n$ to $\{+1, -1\}$ of the form $f(x) = x_i$ for i in $[n]$.

   Consider the following test for whether a function f is a dictator: Given parameter δ, the test chooses $x, y, z \in \{1, -1\}^n$ by first choosing x, y uniformly from $\{1, -1\}^n$, next choosing w by setting each bit w_i to -1 with probability δ and $+1$ with probability $1 - \delta$ (independently for each i), and finally setting z to be $x \circ y \circ w$, where $\circ$ denotes the bitwise multiply operation. Finally, the test accepts if $f(x)f(y)f(z) = 1$ and rejects otherwise.

   (a) Show that the probability that the test accepts is $\frac{1}{2} + \frac{1}{2} \sum_{s \subseteq [n]} (1 - 2\delta)^{|S|} \hat{f}(S)^3$.

(b) Show that if f is a dictator function, then f passes with probability at least $1 - \delta$.

(c) Show that if f passes with probability at least $1 - \epsilon$ then there is some S such that $\hat{f}(S)$ is at least $1 - 2\epsilon$ and such that f is ϵ-close to χ_S.

(d) Why isn't this enough to give a dictator test? (i.e., what non-dictators might pass?) Give a simple fix.

3. **Complexity of Confident Learning:** Show that if there is a PAC learning algorithm for a class C with running time $poly(\log n, 1/\varepsilon, 1/\delta)$, then there is a PAC learning algorithm for C with running time dependence on δ (the confidence parameter) that is only $\log 1/\delta$ – i.e., the "new" PAC algorithm should have run-time $poly(\log n, 1/\varepsilon, log 1/\delta)$. You can assume that the learning algorithm is over the uniform distribution on inputs, although the claim is true in general.

   Note that, in general, the sample complexity is upper bounded by the running time.

4. **Occam Learning of Decision Lists:** We are given a function $f : \{0,1\}^n \to \{0,1\}$ which is known to be a *decision list* (a special case of the decision trees we studied in class): Concretely, f must be constructed from a sequence of decisions, each of which relies on only one literal (see Figure 1).

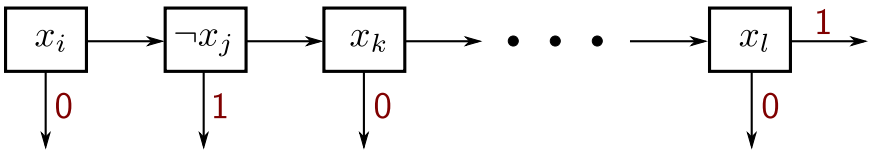

Figure 1: A sample decision list function.

(a) Starting with a set of samples S show how to find a *consistent* decision list h, such that for all $\mathbf{x} \in S$, we have $h(\mathbf{x}) = f(\mathbf{x})$. Note that there may be more than one decision list that is consistent with the data, but proof of Occam's razor shows that all *consistent* decision lists must be close to f. What value of $|S|$ is necessary to ensure (ϵ, δ) PAC learning?

(b) Present an *efficient* (runtime polynomial in n) algorithm to find a decision list h such that, with probability at least $1 - \delta$:

$$\mathbb{P}_{\mathbf{x} \sim \mathcal{D}} [f(\mathbf{x}) \neq h(\mathbf{x})] < \epsilon$$

Hint: Start with an empty decision list, and gradually add more literals, along with their *consequences* (the red output).