CS880: Advanced Learning Theory **Fall 2019**

Lectures 1 & 2: Introduction

Lecturer: Ilias Diakonikolas *Scribes: Ilias Diakonikolas*

1 Class Overview

- Administrative details: See class webpage
- Introductory Content: Unsupervised vs Supervised Learning and Examples. Introduction to Non-parametric Estimation.

In this class, we will study learning and inference from a rigorous mathematical standpoint. Our focus will be on the design and analysis of statistically and computationally efficient learning and inference algorithms in a variety of well-defined models. A rough classification of learning tasks is the following:

1. Unsupervised Learning: Here the goal is to discover hidden structure in a set of *unlabeled* data points, e.g., a set of points in $\mathbb{R}^n$ without corresponding labels. Examples of unsupervised learning include clustering, parameter estimation, density estimation, etc.
2. Supervised Learning: The goal of supervised learning is to make predictions based on a collection of labeled observations, e.g., a set of points in $\mathbb{R}^n$ with a corresponding label for each point. Examples of supervised learning include binary classification, regression, etc.

It should be noted that there are several other learning modes in addition to the ones above, e.g., semi-supervised learning, etc. We will see some additional models later in this class. In the following sections, we provide a concrete definition for two prototypical examples of supervised and unsupervised learning respectively.

2 Supervised Learning Example: PAC Learning

2.1 PAC Learning Definition

The PAC (Probably Approximately Correct) learning model is a classical mathematical model for supervised learning of Boolean functions. It was defined by Leslie Valiant in

the early 80s and is fairly similar to a learning model previously defined by Vapnik.

Let X be the domain of the Boolean functions (binary classification rules) we are interested in. Suppose there exists an unknown function $f : X \to \{0, 1\}$ and a fixed but unknown distribution D supported on X. Given a set of unlabeled examples drawn according to D and their corresponding labels according to f, the goal is to approximately reconstruct f. A bit more formally:

1. Input: a multiset of i.i.d. labeled examples $\{(x^{(i)}, f(x^{(i)}))\}_{i=1}^{m}$, where $x^{(i)} \sim D$ and the $x^{(i)}$ are mutually independent.

2. Output: A hypothesis function $h : X \to \{0, 1\}$ that with high probability is "close" to f in the sense that $\Pr_{x \sim D}[f(x) \neq h(x)] \leq \epsilon$;

Some comments are in order: First, any guarantee that we could hope for should be probabilistic. (This is why we require the "high probability" qualification in 2 above.) Second, the definition as given above is incomplete. Without any a priori assumptions about f, it is easy to see that learning is information-theoretically impossible. A correct definition should specify a family of functions (concept class) $\mathcal{C}$ and the additional assumption that $f \in \mathcal{C}$. Then the complexity of learning becomes a function of the concept class $\mathcal{C}$.

The two most common criteria to evaluate the performance of a learning algorithm are the following:

1. Sample Complexity: How many samples are necessary and sufficient for learning? This is an information-theoretic quantity that in many cases can be precisely characterized. Note that there is a distinction between the sample complexity of a given algorithm ("How many samples does this algorithm need?") and the sample complexity of the corresponding learning problem ("How many samples are needed to solve this learning problem?")

2. Computational Complexity: As usual in algorithms, we care about the runtime of our algorithms, which is measured as a function of the input size (set of samples). Again there is a distinction between the computational complexity of a given algorithm and the computational complexity of a learning problem. In many cases, the computational complexity of a learning problem is trickier to pin-down than its sample complexity.

The field of statistics historically focused on characterizing the sample complexity of learning/inference tasks. The computational complexity considerations have become of paramount importance in recent decades and are typically studied within computer

science. In addition to sample and computational complexity, there exist various other performance criteria (robustness to model misspecification, privacy, communication, memory) some of which we will study in detail later in the class.

2.2 PAC Learning Example: Learning LTFs

One of the most important concept classes in machine learning is the class of Linear Threshold Functions (LTFs) or Halfspaces. LTFs correspond to a basic classification rule where the "positive" examples and the "negative" examples are linearly separable, i.e., are separated by a hyperplane. More formally:

Definition 1 (Linear Threshold Function)**.** *An LTF over $X \subseteq \mathbb{R}^d$ is any Boolean-valued function of the form $f(x) = \text{sign}(w \cdot x - \theta)$. Here* sign() *denotes the "sign" function, defined by* $\text{sign}(t) = 1$ *if* $t \geq 0$*, and* $\text{sign}(t) = -1$ *otherwise. The vector $w \in \mathbb{R}^d$ is called the "weight vector" and the parameter θ is called the "threshold".*

Let us try to come up with a PAC learning algorithm for LTFs over $X = \mathbb{R}^d$. Let $f(x) = \text{sign}(w^* \cdot x - \theta^*)$ be the unknown target LTF. Roughly speaking, our goal is to approximate the correct parameters w^* and θ^*. Specifically, we want to find parameters w' and θ' such that the hypothesis LTF $h(x) = \text{sign}(w' \cdot x - \theta')$ is close to f, i.e., with high probability satisfies $\Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$.

Each labeled example $(x^{(i)}, y_i)$ gives rise to an inequality of the form $y_i(w \cdot x^{(i)} - \theta) \geq 0$ in variables w, θ. Note that each such inequality is satisfied by the parameters w^*, θ^* of the unknown target LTF.

Given the above observation, a natural approach would be to form a system of inequalities, one for each of our m labeled examples, and then solve that system for w, θ. Note that each inequality is linear in its variables w, θ. That is, the corresponding system of inequalities is a *linear program*. Also note that this linear program is *feasible*, as each inequality is satisfied by w^*, θ^*.

It is well-known that the computational problem of linear programming (LP) is in P, i.e., there exists a polynomial-time algorithm to solve it. (One could use either the ellipsoid method or interior-point methods, but this is not relevant in our discussion. We will be using a polynomial time algorithm for LP as a black-box.)

So, suppose that we take m labeled examples from f and then solve for the corresponding LP. This is a computationally efficient algorithm, but does it actually work? The answer turns out to be "YES", assuming m is large enough, but proving such a fact requires the notion of VC dimension and Occam's Razor/Empirical Risk Minimization (ERM). These are very basic tools in PAC learning that we will review in a subsequent lecture. In more detail, if $m = \Omega(d/\epsilon)$, it turns out that any feasible

solution to the corresponding LP gives rise to a hypothesis LTF h that with high constant probability satisfies $\Pr_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$.

Finally, we note that the PAC learning algorithm for LTFs described above outputs an LTF as its hypothesis. These types of learning algorithms, where the hypothesis function also belongs to the concept class $\mathcal{C}$ are called *proper*. In general, it is not necessary that a learning algorithm is proper. Moreover, there are natural learning problems where proper learning is computationally hard, but computationally efficient non-proper learning algorithms exist.

3 Unsupervised Learning Example: Nonparametric Inference

3.1 Non-parametric Learning/Estimation

There are several different formulations of unsupervised learning modeling various phenomena. Here we focus on the nonparametric estimation setting and define three related statistical tasks.

The general setup is as follows: Let $\mathcal{D}$ be a family of probability distributions. For example, $\mathcal{D}$ could be the family of all discrete distributions on $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. There is a fixed but unknown distribution $p \in \mathcal{D}$. Given sample access to p, the goal is to "learn some information" about p. More concretely, we will consider the following three tasks of decreasing difficulty:

- Density Estimation (Distribution Learning): Here the goal is to approximate p itself, i.e., find a hypothesis distribution h such that $h \approx p$. The notion of approximation is typically defined using a "metric", i.e., a function d that quantifies the distance between probability distributions. Typically, we are given an accuracy parameter $\epsilon > 0$ and we want to guarantee that $d(h, p) \leq \epsilon$ with high probability.

- Distance Estimation/Tolerant Testing: There are a few essentially equivalent definitions of this task. One of them is as follows: Given a "property" $\mathcal{P}$, i.e., a set $\mathcal{P} \subseteq \mathcal{D}$, the goal is to approximate the distance of p from $\mathcal{P}$, defined as the minimum distance between p and any $q \in \mathcal{P}$: $d(p, \mathcal{P}) = \inf_{q\in\mathcal{P}} d(p, q)$.

- Hypothesis Testing (Distribution Testing): Given a "property" $\mathcal{P}$ the goal is to correctly distinguish, with high probability, between the case that $p \in \mathcal{P}$ and the case that $d(p, \mathcal{P}) \geq \epsilon$.

It is important to note that the underlying distribution family $\mathcal{D}$ and the choice of the particular metric d can make a big difference in terms of the appropriate learning/estimation/testing algorithm and the corresponding sample/computational complexity of the problem.

As in the case of supervised learning, we will be primarily be interested in the sample and computational complexities of these problems. Moreover, we will explore additional criteria, including model misspecification, privacy, communication, etc.

In the following subsection, we give a concrete example of distribution learning for unstructured discrete distributions under the total variation distance metric.

3.2 Learning Discrete Distributions

In this section, we will study the problem of learning discrete distributions over $[n]$ under the total variation distance metric.

For notational convenience, we will identify a probability distribution on $[n]$ with the corresponding probability mass function p that we view as a vector $p = (p_1, \ldots, p_n)$. That is, we will take $\mathcal{D} \stackrel{\text{def}}{=} \{p = (p_1, \ldots, p_n) \mid p_i \geq 0, \sum_i p_i = 1\}$.

The total variation distance between two distributions $p, q : [n] \to [0, 1]$ is defined as:

$$d_{\mathrm{TV}}(p, q) \stackrel{\text{def}}{=} \max_{S \subseteq [n]} |p(S) - q(S)| = (1/2)\|p - q\|_1 \ ,$$

where $p(S) \stackrel{\text{def}}{=} \sum_{i \in S} p_i$ and $\|v\|_1$ denotes the ℓ_1-norm of the vector v. Note that the first equality is the actual definition. The second equality is an identity, a corollary of the definition, and is a simple exercise.

With the above definitions in place, the density estimation task is the following:

> Given access to iid samples drawn from an unknown distribution $p \in \mathcal{D}$, output a hypothesis distribution h that with probability at least $1 - \delta$ satisfies $d_{\mathrm{TV}}(h, p) \leq \epsilon$.

The sample and computational complexity of this task will be a function of the domain size n, the desired accuracy $\epsilon > 0$, and the failure probability $\delta > 0$. Ideally, we would like to design a learning algorithm with information-theoretically optimal sample complexity (as a function of all three parameters), that runs in (near-)linear time. For this basic problem, we will see that this goal is attainable.

Learning Algorithm. The learning algorithm in this case is very simple: We draw a number of samples and estimate each probability p_i by its "empirical frequency". The vector of empirical frequencies is sometimes called the "empirical distribution".

The pseudo-code for the algorithm follows:

Algorithm Discrete Density Estimation
Input: sample access to a distribution p over $[n]$, $\epsilon, \delta > 0$.
Output: a distribution h such that $d_{\mathrm{TV}}(h, p) \leq \epsilon$ with probability $1 - \delta$.

1. Draw m samples s_j, $j \in [m]$, iid from p.
2. For $i \in [n]$ and $j \in [m]$, let X_{ij} be the indicator of the event that s_j equals i.
3. For $i \in [n]$, define $h_i = \frac{\sum_{j=1}^{m} X_{ij}}{m}$.
4. Output h.

In the rest of this section, we analyze this algorithm.

Analysis. We want to show that with high probability over the samples drawn from p the total variation distance $d_{\mathrm{TV}}(h, p)$ is small. A natural way to proceed is by analyzing the expectation of the random variable $d_{\mathrm{TV}}(h, p)$. This is done in the following lemma:

Lemma 2. *Let h be the hypothesis distribution output by the above algorithm. Then we have that $\mathbf{E}[d_{\mathrm{TV}}(h, p)] \leq O(\sqrt{\frac{n}{m}})$.*

Proof. First, we note that $d_{\mathrm{TV}}(h, p)$ is a random variable, as it is a function of h, which itself is as a function of the samples drawn from p. By definition, we have that $d_{\mathrm{TV}}(h, p) = (1/2) \sum_{i=1}^{n} |p_i - h_i|$. By taking expectations of both sides and using linearity of expectation, we get:

$$\mathbf{E}[d_{\mathrm{TV}}(h, p)] = (1/2) \sum_{i=1}^{n} \mathbf{E}[|p_i - h_i|] \; . \tag{1}$$

To bound (1) from above, we need to get a handle on a generic term in the sum.

By definition of h, we have that $m \cdot h_i = \sum_{j=1}^{m} X_{ij}$. Note that for each fixed $i \in [n]$, X_{ij} is a Bernoulli (0/1) random variable with expectation p_i and that the X_{ij}, $j \in [m]$ are independent. That is, $\sum_{j=1}^{m} X_{ij}$ is a Binomial random variable with parameters m and p_i, i.e., $m \cdot h_i \sim \mathrm{Bin}(m, p_i)$. Consequently, we have that

$$\mathbf{E}[h_i] = (1/m)\mathbf{E}[\mathrm{Bin}(m, p_i)] = (1/m) \cdot m p_i = p_i \; ,$$

and

$$\mathbf{Var}[h_i] = (1/m)^2 \cdot \mathbf{Var}[\mathrm{Bin}(m, p_i)] = (1/m)^2 \cdot mp_i(1-p_i) \leq (1/m)p_i \ .$$

Recall that for any squarely integrable random variable Z, we have that $\mathbf{E}[Z] \leq \sqrt{\mathbf{E}[Z^2]}$. This is a consequence of convexity and Jensen's inequality. Applying this inequality to the random variable $|h_i - p_i|$, we get:

$$\mathbf{E}[|h_i - p_i|] \leq \sqrt{\mathbf{E}[(h_i - p_i)^2]} = \sqrt{\mathbf{Var}[h_i]} \leq \sqrt{p_i/m} \ , \tag{2}$$

where the equality follows from the fact that $\mathbf{E}[h_i] = p_i$.

By combining (1) and (2), we obtain that:

$$\mathbf{E}[d_{\mathrm{TV}}(h,p)] \leq (1/2)\sum_{i=1}^{n}\sqrt{p_i/m} = (1/2)\frac{\sum_{i=1}^{n}\sqrt{p_i}}{m} \ .$$

To bound the RHS above, we note that $\sum_{i=1}^{n}\sqrt{p_i}$, subject to the constraint that $\sum_{i=1}^{n} p_i = 1$, is maximized when $p_i = 1/n$, for all $i \in [n]$, with maximum value $\sqrt{n}$. There are various ways to show this. Perhaps the easiest one is to apply the probabilistic version of Jensen's inequality to the concave function $g(x) = \sqrt{x}$.

In summary, we have shown that

$$\mathbf{E}[d_{\mathrm{TV}}(h,p)] \leq (1/2)\sqrt{\frac{n}{m}} \ .$$

This completes the proof of the lemma. □

Give our bound on the expectation of the error, we can obtain a high-probability bound by a simple application of Markov's inequality.

Proposition 3. *For $m = \Omega(n/\epsilon^2)$, with probability at least 9/10 over the samples drawn from p, we have that $d_{\mathrm{TV}}(h,p) \leq \epsilon$.*

Proof. Since $d_{\mathrm{TV}}(h,p)$ is non-negative, the proposition follows as a corollary of the previous lemma, by an application of Markov's inequality. □

It can be shown that the above sample complexity upper bound is information-theoretically best possible, within constant factors, when the desired failure probability δ is assumed to be a constant.

On the other hand, when δ is viewed as a parameter (which could potentially be very small), the same Markov argument above leads to a sample complexity bound

$m = \Omega((1/\delta) \cdot n/\epsilon^2)$. This linear dependence on $1/\delta$ is very sub-optimal. It turns out that the optimal sample complexity of this algorithm (and the learning problem itself) is $\Theta((n+\log(1/\delta))/\epsilon^2)$, i.e., it is logarithmic in $1/\delta$ and moreover is additive, as opposed to multiplicative.

To show that our proposed algorithm achieves this optimal sample complexity bound (as a function of all parameters), we require a different argument. While there are several equivalent ways to show this, perhaps the simplest proof makes use of the Chernoff bound and the union bound. The tricky part is that one needs to use the actual definition of the total variation distance, i.e., $d_{\mathrm{TV}}(p,h) \stackrel{\mathrm{def}}{=} \max_{S \subseteq [n]} |p(S) - h(S)|$. This is left as an easy exercise.

Sample Complexity Lower Bound So far, we have described a simple way to derive an *upper bound* on the sample complexity of learning discrete distributions. It turns out there is a rich theory in statistics and information-theory to establish *unconditional sample complexity lower bounds*. In our specific setting, it can be shown that *any* algorithm that learns an arbitrary discrete distribution on $[n]$ within total variation distance ϵ with probability at least $1-\delta$ requires at least $\Omega((n+\log(1/\delta))/\epsilon^2)$ samples.

We will give a detailed proof of this fact in a subsequent lecture, but for now let us intuitively explain why the dependence on n and ϵ is the right one. For the dependence on n, we claim that even for constant values of ϵ and δ, any algorithm should use at least $\Omega(n)$ samples. Roughly speaking, this holds because we need to at least see all domain elements in the effective support of our distribution.

To see that the dependence on $1/\epsilon$ ought to be at least quadratic, we note that the $n = 2$ special case of our problem captures the problem of distinguishing between a fair coin and an ϵ-biased coin. The latter is a very basic problem in information-theory that has sample complexity of $\Theta(\log(1/\delta)/\epsilon^2)$ for failure probability of δ.

4 Testing Properties of Discrete Distributions

The problem of learning a discrete distribution can be solved with a number of samples linear in the domain size, for constant accuracy and confidence parameters. While this dependence on the domain size is unavoidable for the problem of learning, typical hypothesis testing tasks can be solved with significantly fewer samples.

We have already defined the generic distribution property testing question. Here are some examples of fundamental distribution testing tasks that we will study in this course:

- Uniformity Testing: Test whether an unknown distribution p on a discrete domain is the uniform distribution on its domain versus far from being uniform.
- Identity Testing (Goodness-of-fit): Test whether an unknown distribution p is equal to an explicitly given distribution q versus far from q.
- Equivalence/Closeness Testing (Two-sample testing): Test whether two unknown distributions p, q are equal versus far from each other.
- Independence Testing: Test whether a distribution over tuples is a product distribution versus far from any product distribution.
- Testing Shape Constraints: Test whether an unknown distribution satisfies some shape constraint (e.g., monotonicity, log-concavity) versus being far from satisfying the constraint.

In the following lecture, we will give an algorithm for uniformity testing under the total variation distance whose sample complexity scales with the *square root* of the domain size.