Lecture 5: November 12

Lecturer: Amos Fiat *Scribe: Eran Werner and Ben Sandbank*

5.1 Lecture Overview

In this lecture we reintroduce the concept of PAC learning, and show how the selection of different hypothesis-sets influences the learning algorithm's performance. We do this by presenting a problem that is not *efficiently* PAC-learnable using one set of concepts, but is efficiently PAC-learnable using another. Finally, we shortly discuss the issue of *Occam Learning* and give a theorem relating it to PAC-Learning.

5.2 PAC-Learning

5.2.1 Preliminaries

Let us review the definition of PAC-Learning.

Definition Let X be the space in question (R^n, $\{0,1\}^n$, etc'). Let C and H be sets of concepts, not necessarily equal(a concept, as always, being a subset of X). An algorithm A is said to be a *PAC-Learning algorithm* if, getting learning examples selected from X according to some probability distribution D, for any $c \in C, \delta > 0, \epsilon > 0$ the algorithm returns a concept $h \in H$ such that in probability larger than $(1-\delta)$ it holds that -

$$Pr_D(h(x) \neq c(x)) < \epsilon. \tag{5.1}$$

More intuitively, a PAC-Learning algorithm is an algorithm that given a concept to be learned, returns a concept that with high probability is very similar to it (hence its name - 'Probably Approximately Correct'). The input to the algorithm is simply samples from the space X, assumedly as many as the algorithm requires (the number of samples available for the algorithm is not one of its parameters). The 'similarity' of the returned hypothesis only holds in context of the distribution D, the distribution according to which the learning samples were selected.

Definition A PAC-Learning algorithm is said to be *efficient* if its running time is polynomial in $1/\delta$, $1/\epsilon$, $size(c)$.

What is meant by $size(c)$? This is the number of bits it takes to encode the concept c.

Obviously, this size depends on the encoding we use for concepts, so efficiency depends on the language used to describe the concepts.

5.2.2 Hypothesis class

In the definition of PAC-Learning the class of concepts potentially returned by the algorithm (H) is not necessarily the same as the class of concepts used for the learning tasks (C). But is there any reason for this difference? Wouldn't it be sufficient to use the same class for both purposes? Turns out the answer is no. Following we will give an example of a set of concepts that isn't efficiently PAC-Learnable when $H = C$, but is when another set of hypotheses is chosen.

Theorem 5.1 *Let $X = \{0,1\}^n$, and let* C *be the set of concepts consisting of 3-term DNF formulas (a 3-term DNF formula is a logical formula of the form $(x_{i_1} \wedge x_{i_2} \wedge ... \wedge x_{i_I}) \vee (y_{j_1} \wedge y_{j_2} \wedge ... \wedge y_{j_J}) \vee (z_{k_1} \wedge ... \wedge z_{k_K})$ - Each binary vector is treated as a logical assignment of n-logical variables. A vector is said to* belong *to a concept iff the corresponding assignment satisfies the corresponding formula), then if there exists an efficient PAC-Learning algorithm that uses $H = C$ then $P = NP$.*

Proof: In the following we will show that if there is such an algorithm as stated in the theorem, then it is possible to use it in order to solve the NP-Complete problem of determining whether a graph is 3-colorable. Let's restate that more formally.

Definition

Let *G-3COL* be the set of all 3-colorable graphs. Our claim can be stated as follows

$$\alpha \in G - 3COL \leftrightarrow \exists S_\alpha \subseteq X \times \{0,1\} \textit{consistent with some concept } c \in C \tag{5.2}$$

This claim means that for each 3-colorable graph there is some subset of samples from X, which is consistent with some 3-term DNF formula, and vice-versa. This consistency is something that can be checked using a PAC-Learning algorithm - take D to be the uniform distribution over S^+. Then take $\epsilon \leq \frac{1}{2|S^+|}$. With probability as high as we please the algorithm will return (in polynomial time, assuming its efficiency) the correct concept iff the corresponding graph is 3-colorable. If it is not, then the algorithm will either return an incorrect hypothesis, or will run for longer than he should otherwise - both events can be detected in polynomial time. The conclusion will be, naturally, that $P = NP$.

What remains to be seen is the transformation that takes a 3-colorable graph and returns a classified subset of X, subject to the claim above.

Definition

Let us now assume that $G = (V, E)$ is a 3-colorable graph. We need to show that $S_G = S_G^+ \cup S_G^-$ is consistent with a 3-term DNF c. Indeed, let's take $c = T_R \cup T_B \cup T_Y$, where

- $T_R = \bigcap_{v_j notRed} x_j$;
- $T_B = \bigcap_{v_j notBlue} x_j$;
- $T_Y = \bigcap_{v_j notYellow} x_j$

in some legal coloring of G (with red, blue and yellow being the three colors used). Let's see if S_G is consistent with this concept. Firstly, for each $s \in S_G^+$ there is a corresponding node v_j, such that $s_j = 0$ while $s_i = 1$ for all $i \neq j$. This x_j does not appear in one of the clauses (that corresponding to its color in a legal coloring), and all the other literals = 1 - hence the clause's value is true, and the sample is classified 1 by the concept. Similarly, the concept classifies each negative sample in S_G as 0. This is because each such sample corresponds to a node (v_i, v_j), being 0 in the i'th and j'th coordinates, 1 otherwise. Since each node is colored in a different color (from the definition of legal coloring), each clause will contain at least one of the corresponding $x_i or x_j$, and hence be false. What remains to be proven is the opposite direction - assume that $G = (V, E)$ is a graph, and S_G is consistent with a 3-DNF formula. We need to show that G is 3-colorable. The formula contains three clauses, for each we will assign a different color. For each node in V we will assign the color of the first clause in the formula that the corresponding vector satisfies. Now let $e = (v_i, v_j) \in E$, and assume v_i and v_j were colored with the same color, then both of the corresponding vectors satisfy the same clause in the 3-DNF formula. By our definition of the corresponding vectors, it follows that x_i and x_j do not appear in that clause. However, in that case the vector - $<(11...101...101...1)>$, with zeros in the i'th and j'th places, also satisfies the clause, which means it satisfies the whole formula - i.e. classified positively by the concept - contradiction. $\square$

We shall now go on to seeing how choosing a different set of hypotheses, namely that of all 3-CNF formulas, allows for an efficient PAC-Learning algorithm for the same set of concepts. Note that the set of 3-CNF formulas contains and is not equal to the set of 3-term DNF formulas. Each 3-term DNF formula is equivalent to a 3-CNF formula, but the reverse is not true.

First we'll present an efficient PAC-Learning algorithm for the concept set consisting of all conjunctions (formulas of the form $l_1 \wedge l_2 \wedge ... \wedge l_n$ where l_i is a literal - either a variable or its negation).

The learning algorithm is very simple. It will ignore all negative samples, making use of only the positive ones. Throughout the learning period the algorithm will maintain the *'current clause'*, reflecting the information gathered thus far in the learning process. At first, this clause is simply - $x_1 \wedge \overline{x}_1 \wedge x_2 \wedge \overline{x}_2 \wedge ... \wedge x_n \wedge \overline{x}_n$. For each positive sample, the algorithm will remove from the current clause the *negations of the literals appearing in the sample formula*. For example, if a positive sample contains the literal x_i we know for sure that the concept to be learned does not contain the literal $\overline{x}_i$.

Let $\epsilon > 0$, $\delta > 0$. We want to estimate the number of samples needed by the algorithm to return a concept h that at probability $> (1-\delta)$ has $error(h) < \epsilon$. If this number is polynomial in $1/\epsilon$ and $1/\delta$ then we have proven our claim, since each sample is processed in $O(n)$ time.

First, note that the algorithm never errs on positive samples. Therefore, if we write for every literal z -

$$p(z) = Prob_{a \in D}(c(a) = 1 \wedge z = 0 ina) \tag{5.3}$$

then -

$$error(h) \leq \sum_{z \in h} p(z) \tag{5.4}$$

Definition We'll call a literal z *bad* if $p(z) \geq \frac{\epsilon}{2n}$.

If h contains no *bad* literals, then we'll conclude that $error(h) < \epsilon$. Let z be a *bad* literal. The probability that z appears in the algorithm's 'current clause' after m samples have been processed is, from the definition of bad literals, $(1 - \frac{\epsilon}{2n})^m$. Given there are $2n$ literals all in all, we can derive an upper bound on the probability that any bad literal remains in the algorithm's hypothesis after m samples - $2n(1-\frac{\epsilon}{2n})^m$. In order for the algorithm to be a PAC-learning algorithm, this probability has to be smaller than δ. From this we can derive the number of samples needed by the algorithm -

$$2n(1 - \frac{\epsilon}{2n})^m < \delta \Rightarrow m \geq (\frac{2n}{\epsilon})(\lg(2n) + \lg(\frac{1}{\delta})) \tag{5.5}$$

.

m is indeed polynomial in n, $1/\epsilon$ and $1/\delta$, and the presented algorithm is therefore an efficient PAC-Learning algorithm for this concept set.

As it turns out, the very same algorithm can be used for learning 3-CNF formulas. The trick is to define for each clause in the 3-CNF formula a single variable. Thus we create a new space where the concepts are equivalent to conjunction-formulas. Each sample in the original space can be converted into a sample in the new space, and used to drive the PAC-Learning algorithm just presented. Since the number of variables in the new space is $O(n^3)$, it is still an efficient algorithm.

5.3 Occam Learning

Except for PAC-Learning, which has been attracting growing attention in recent years, there are other schemes of computational learning. Among the most important of these is the scheme called *Occam Learning*.

Definition Let X be an n-dimensional space, and let $\{<\overline{x}_1, c_1>, ..., <\overline{x}_m, c_m>\}$ be a set of m classified samples. Also, let $H_{n,m}$ be a set of possible hypotheses. An algorithm is an Occam-learning algorithm if it finds a hypothesis $h \in H_{n,m}$ which -

- is consistent on all samples; and
- $\lg |h| \leq m$, $|h|$ being the description length of hypothesis h.

Intuitively, Occam-learning algorithms find a hypothesis that compresses the data contained in the classification of the samples to less than the number of bits needed to represent this classification in the trivial way. Moreover, the more efficient the compression, so the algorithm performs better (a better compression ignores more redundant data in the samples, and therefore is more likely to generalize better on other, unseen samples).

Note that contrary to the PAC-learning scheme, in this scheme the number of samples provided to the algorithm is predetermined, and not up to the algorithm's discretion. This makes this model very well suited for data-mining purposes, where the number of samples is not necessarily limitless and good generalization is still desired. As it turns out, however, there is a link between Occam-learning and PAC-learning. If an effective Occam-learning algorithm exists for a task, so there is an efficient PAC-learning algorithm for that task as well. More formally -

Theorem 5.2 *Let $H_{n,m}$ be as above, and assume the existence of an Occam learning algorithm for the given task. Let $\epsilon > 0$, $\delta > 0$, and let D be some probability distribution over X. Also let*

$$m \geq \frac{1}{\epsilon}(\lg(|H_{n,m}|) + \lg(\frac{1}{\delta})). \tag{5.6}$$

Then if we take m samples according to the distribution D and pass them to the Occam-learning algorithm, the hypothesis it will return will have $error(h) < \epsilon$ with probability greater than $(1 - \delta)$. In other words, we will have an efficient PAC-learning algorithm.

The proof for this theorem is beyond the scope of this lecture.

Obviously, this theorem is only useful for the cases where $|H_{n,m}|$ is finite. For example, if we take as our set of possible hypotheses the set of all decision lists, $|H_{n,m}| = O(n!)$, and $\lg(|H_{n,m}|) = O(n \lg n)$. For the cases where $|H_{n,m}|$ is infinite, we need to make use of the *VC dimension* of the hypothesis set. This will elaborate on in the next lecture...