Computational Learning Theory Lecture Notes

Martin Citoler-Saumell

CIS625 Spring 2017

These are notes from Prof. Michael Kearns's CIS625. This course is currently being taught at the University of Pennsylvania and attempts to provide algorithmic, complexity-theoretic and probabilistic foundations to modern machine learning and related topics. You can find out more at the course website. A word of caution, it is likely that some typos and/or errors have been introduced during the taking of these notes. If you find any, please let me know at martinci@math.upenn.edu.

Contents

1 Lecture 1: 2017.01.23

1.1 Course Outline/Description

1.1.1 Formal Models of ML

- Assumptions about data generation process.
- Assumptions about what algorithms "knows".
- Sources of info that the algorithms has
- Criteria/objective of learning is.
- Restrictions on algorithms.

1.1.2 Examples of Models

- "PAC" model (in first 1/2 of the term).
- Statistical learning theory.
- "no-regrets" learning models.
- Reinforcement learning.
- ML & Differential privacy.
- "Fairness" in ML

1.2 A Rectangle Learning Problem

Suppose you are trying to teach an alien friend the "shape" of humans in terms of abstract descriptions like "medium build", "athletic", etc. We are going to assume that each one of these descriptions represents a rectangular region on the height-weight plane but we are not aware of the exact dimensions. The only thing we are able to tell the alien is whether a particular individual is medium built or not. i.e. we can only label examples.

- target rectangle R, the *true* notion of "medium built".
- hypothesis rectangle $\hat{R}$.

Remark 1.1. Note that the assumption that the classifier function is a rectangle is rather strong. There is always this trade-off to be able to actually compute things. From a Bayesian point of view, "we always need a prior".

Given a data cloud of examples, a reasonable hypothesis rectangle could be the tightest fit rectangle. However, this choice ignores the negative examples so it seems that that we are throwing away information. In a sense, this rectangle would be the least likely.

- Assume $\langle x_1, x_2 \rangle$ pairs of height-weight are i.i.d. from an arbitrary unknown probability distribution, D.

We want to be able to evaluate how our hypothesis rectangle is performing. We want bounds on the classification error, which can be thought as the size of the symmetric difference between R and $\hat{R}$

$$D[R \triangle \hat{R}] \equiv \mathbb{P}_D[R(\vec{x}) \neq \hat{R}(\vec{x})]. \tag{1.1}$$

Theorem 1.2. *Given $\varepsilon, \delta > 0$, there is some integer N such that if we have more than N training examples,*[1] *we have*

$$D[R \triangle \hat{R}] \equiv \mathbb{P}_D[R(\vec{x}) \neq \hat{R}(\vec{x})] < \varepsilon, \tag{1.2}$$

with probability at least $1 - \delta$.

[1] This the same as saying that sample size is at least N.

Proof. First of all, note that using tightest fit, the hypothesis rectangle is always contained inside the target rectangle. Now, for each side of the target rectangle we may draw inward strips in such a way that each strip has $\mathbb{P}_D[Strip] < \frac{\varepsilon}{4}$. If the training set has a positive example in each of these four strips, then the inequality above is satisfied because the boundary of the hypothesis rectangle would be contained in the union of the strips. Next we need to deal with the required sample size to obtain this result with some certainty. Let m denote the sample size, since the distribution is i.i.d., we have

$$\mathbb{P}_D[\text{miss a specific strip m times}] = \left(1 - \frac{\varepsilon}{4}\right)^m,$$
$$\mathbb{P}_D[\text{miss any of the strips m times}] \geq 4\left(1 - \frac{\varepsilon}{4}\right)^m.$$

By the discussion above, the last inequality implies

$$\mathbb{P}_{D^m}[D[R \triangle \hat{R}] \geq \varepsilon] \leq 4\left(1 - \frac{\varepsilon}{4}\right)^m,$$

which can be chosen arbitrarily small for big enough m. One can obtain $N \geq \frac{4}{\varepsilon}\ln\left(\frac{4}{\delta}\right)$. □

Remark 1.3. This proof generalizes to d-dimensional rectangles. We only need to replace 4 with $2d$, the number of $(d-1)$-faces. We can also try to incorporate noisy data, where the labels have some probability of being wrong.

1.3 A More General Model

- Input/instance/feature space, X. (e.g. $\mathbb{R}^2$ in the example above)
- Concept/classifier/boolean function, $C : X \to \{0, 1\}$ or we can also think about it as an indicator function of the positive examples or the subset of positive examples.
- Concept class/target class, $\mathcal{C} \subset \mathcal{P}(X)$, the admissible concepts/classifiers. (e.g. all rectangles in $\mathbb{R}^2$ in the example above)
- target concept, $c \in \mathcal{C}$. (e.g. target rectangle in the example above)
- Input distribution, D over X (arbitrary & unknown)
- Learning algorithm given access to examples of the form: $\langle \vec{x}, y \rangle$ where $\vec{x}$ is i.i.d. drawn from D and $c(\vec{x}) = y$.

Definition 1.4 (PAC Learning)**.** We say that a class of functions over X, $\mathcal{C}$, is *Probably Approximately Correct (PAC) learnable* if there exists an algorithm, L, such that given any c in $\mathcal{C}$, D a distribution over X and $\varepsilon, \delta > 0$, L with these parameters and random inputs $\vec{x}$'s satisfies:

- (Learning) With probability $\geq 1-\delta$, L outputs a hypothesis, h in $\mathcal{C}$ such that $D[h \triangle c] < \varepsilon$, i. e.

$$Err(h) \coloneqq \mathbb{P}_{x \sim D}[h(x) \neq c(x)] \leq \varepsilon. \tag{1.3}$$

- (Efficient) L runs in time/sample *poly* $\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, \text{dimension}\right)$.

2 Lecture 2: 2017.01.30

Remark 2.1 (PAC Learning). Fixing the class $\mathcal{C}$ is a strong assumption, it is the prior you are assuming about the true behavior of the data. For example, when you fit a linear regression, you are assuming that there is a true linear relation in the data.

In contrast, the assumption on the distribution over X is fairly general.

Theorem 2.2. *The class of rectangles over $\mathbb{R}^2$ from example above is PAC learnable (with sample size $m \sim \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)$).*

2.1 PAC learning boolean conjunctions

In the following we are going to see an example of problem that is PAC learnable.

- $X = \{0,1\}^n$
- Class $\mathcal{C}$ = all conjunctions over $x_1, \ldots, x_n$. $|\mathcal{C}| = 3^n$
  E.g.: If $c = x_1 \wedge \neg x_3 \wedge \neg x_{11} \wedge x_{26} \ldots$,

$$c(\vec{x}) = 1 \iff x_1 = 1, x_3 = 0, x_{11} = 0, x_{26} = 1, \ldots \tag{2.1}$$

- D over $\{0,1\}^n$.

2.1.1 Algorithm for monotone case (i.e. no $\neg$'s)

In this case we are trying to fit something like $c = x_1 \wedge x_5 \wedge x_{13} \ldots$. i.e. given some examples of sequences of bits and the result of c on them, we are trying to guess what the conjunction c actually is. We can use the following algorithm:

- $h \leftarrow x_1 \wedge x_2 \wedge x_3 \wedge \ldots \wedge x_n$, start with the conjunction of all the variables.
- For each positive example, $\langle \vec{x}, 1 \rangle$, delete any variable in h such that $x_i = 0$.
  This method ensures that the positives of h is a subset of the true c.

Analysis

Let p_i denote the probability we delete x_i from h in a single draw. In other words,

$$p_i = \mathbb{P}_{\bar{X} \sim D}[c(\vec{x}) = 1, x_i = 0]. \tag{2.2}$$

Then we have an a priori bound on error: $Err(h) \leq \sum_{x_i \in h} p_i$. We can make a distinction between *bad* and *good* indices. An index, i, is bad if $p_i \geq \frac{\varepsilon}{n}$ and good otherwise. Note that if h contains no bad indices, then we have

$$Err(h) \leq \sum_{x_i \in h} p_i \leq n \left(\frac{\varepsilon}{n}\right). \tag{2.3}$$

Let's fix some index, i, such that $p_i \geq \frac{\varepsilon}{n}$. i.e. a bad index. We have that

$$\mathbb{P}[x_i \text{ "survives" m random samples}] = (1-p_i)^m \quad (iid) \tag{2.4}$$

$$\leq \left(1 - \frac{\varepsilon}{n}\right)^m \tag{2.5}$$

$$\mathbb{P}[\textit{any} \text{ bad i "survives" m random samples}] \leq n\left(1 - \frac{\varepsilon}{n}\right)^m. \tag{2.6}$$

If we want the right-hand-side of the last inequality to be less than some $\delta > 0$, we end up with $m \geq \frac{n}{\varepsilon} \ln\left(\frac{n}{\delta}\right)$. In other words, we just proved the following theorem

Theorem 2.3. *Conjunctions over $\{0,1\}^n$ are PAC learnable with sample size $m \geq \frac{n}{\varepsilon} \ln\left(\frac{n}{\delta}\right)$.*

Remark 2.4. An analogous argument proves this theorem for conjunctions that are not necessarily monotone. The only difference is that we have to keep track the extra $\neg$ variables.

Remark 2.5. We can identify some pattern for this kind of analysis. We identify some "bad" things that may happen and then prove that the probability of them happening decreases fast when we increase the number of samples seen.

2.2 Hardness of PAC learning 3-term DNF

Now we are going to see an example of non PAC learnable problem. However, we will be able to slightly modify it and achieve PAC learning. This motivates an better definition of PAC learnable.

- Input space $X = \{0,1\}^n$
- Class $\mathcal{C}$ = all disjunctions of three conjunctions. $|\mathcal{C}| = 3^{3n}$
  E.g.: If $c = T_1 \vee T_2 \vee T_3$ where T_i is a conjunction over X.
  $c = (x_1 \wedge x_7 \wedge \neg x_8) \vee (x_2 \wedge x_5 \wedge \neg x_7 \wedge \neg x_8) \vee (x_6 \wedge x_{12})$.

To see that this problem is *hard*, we prove that the graph 3-coloring problem reduces to the 3-term DNF problem.

Graph 3-coloring to PAC learning 3-term DNF

Suppose that you have a 3-colorable (undirected) graph G. That is, a graph such that we can color the vertices with 3 colors in such a way that there are no edges between vertices of the same color. We see an example of how to transform such a graph into a set of labeled examples for the PAC learning 3-term DNF.

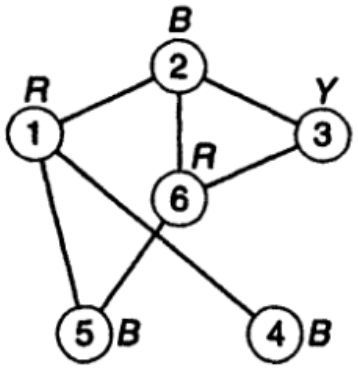


Figure 1: A graph with a 3-coloring.

First, for the i-th node we create a positive example that is represented by the vector, $v(i)$, with a 0 in the i-th entry and 1's everywhere else. E.g. from node one we have $\langle 011111, + \rangle$. Then we create a negative example from each edge that is represented by a vector, $e(i, j)$, of 1's except for 0's at the positions that determine the edge. E.g. from the edge connecting 2 and 3 we obtain $\langle 100111, - \rangle$. Next, the coloring can be used to define a 3-term DNF in the following way. Given a color, we define T_{color} as the conjunction of the variables/vertices that are *not* of that color. In this example we have

$$T_R = x_2 \wedge x_3 \wedge x_4 \wedge x_5, \tag{2.7}$$

$$T_B = x_1 \wedge x_3 \wedge x_6, \tag{2.8}$$

$$T_Y = x_1 \wedge x_2 \wedge x_4 \wedge x_5 \wedge x_6. \tag{2.9}$$

Claim 2.6. *G is 3-colorable $\iff$ there is a 3-term DNF* consistent *with the labeled sample above.*

Proof. We have just seen how to obtain a 3-term DNF from a 3-colorable graph. We only need to check that it is consistent with the sample. By construction, all the positive examples satisfy T_{color} where color is the vertex's color, since the only 0 in the vector is in the position that is dropped. Similarly, it follows that the examples coming from the edges do not satisfy any of the T's. In any edge there are two colors corresponding to its vertices. The extra 0 in the example ensures that the T's from those colors are not satisfied. Finally, the T of the remaining color cannot be satisfied because both vertices are included in the conjunction and they are both 0 in the example.

Conversely, given a graph and the labeled examples as above, if there is a consistent 3-term DNF we can find a coloring in the following way. Label each term of the formula with a color, say $T_R \vee T_Y \vee T_B$, and remember the order of the labels. Then, we define the color of vertex i (corresponding to vector $1 \ldots 101 \ldots 1$) as the label of the first formula that is satisfied by $v(i)$. Since the formula is consistent with the sample, every vertex must be actually colored. We only need to argue that this is a valid coloring. Suppose to the contrary that i and j are to vertices that are connected by an edge and have the same color. This means that both $v(i)$ and $v(j)$ satisfy T_{C_0}. However, we also have $v(i) \& v(j) = e(i, j)$ where $\&$ denotes the bit-wise and operation and it follows that $e(i, j)$ satisfies T_{C_0}, a contradiction with the consistency of the formula since edges are negative examples. $\square$

This concludes the argument that finding a coloring of a graph is the same as producing a consistent 3-term DNF. We are only left with the computational aspects. Namely, given the labeled sample associated to a graph, we need to find a way to feed it to a PAC learning algorithm. First we need a distribution over vectors of bits. For this we can just sample the examples uniformly. Finally, to ensure consistency we choose ε any quantity less than $\frac{1}{\#examples}$[2]. This way the algorithm cannot make any mistakes is forced to be consistent. The δ can be arbitrary. In conclusion, if the 3-term DNF problem were PAC learnable, we could solve the coloring problem in random polynomial time. Another way to say this is in the form of the following theorem.

Theorem 2.7. *If 3-term DNF are PAC learnable, then $NP = RP$.*

Of course, it is strongly believed that $RP \subsetneq NP$ so this is rather strong evidence against the easiness of the 3-term DNF problem.

Remark 2.8. The upshot is that a slight generalization of the conjunction problem, for which we have a randomized polynomial time solution, is almost assured to not be PAC learnable (unless NP = RP)

3 Lecture 3: 2017.02.06

Recall the definition of a class being PAC learnable.

Definition 3.1 (PAC Learning)**.** A class $\mathcal{C}$ is *Probably Approximately Correct (PAC) learnable* if there exists an algorithm, L, such that:

$$\forall c \in \mathcal{C}, \quad \forall D \text{ over } X, \quad \forall \varepsilon, \delta > 0 \tag{3.1}$$

- (Learning) With probability $\geq 1-\delta$, L outputs a hypothesis, h in $\mathcal{C}$ such that $D[h \triangle c] < \varepsilon$, i.e. we have error at most ε

$$Err(h) \coloneqq \mathbb{P}_{x \sim D}[h(x) \neq c(x)] \leq \varepsilon. \tag{3.2}$$

- (Efficient) L runs in time/sample *poly* $\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n\right)$. *Samples & computation.*

Remark 3.2. Usually when we talk about computation time we are in the realm of complexity theory and we talk about samples we are really asking statistics/information-theory questions about what sample size do we need to be able to draw some conclusion.

3.1 PAC learning 3-term DNF by 3-CNF

In the following we see how to overcome the intractability of the 3-DNF PAC learning by using a different representation. It amounts to expanding the input space and the class we are learning.

Definition 3.3. A 3-CNF is a conjunction of disjunctions of length three.

[2]This epsilon is allowed because $\frac{1}{\varepsilon}$ is polynomial in the size of input. This is not true in general.

Given a 3-DNF, we can rewrite it as a 3-CNF in the following way:

$$T_1 \vee T_2 \vee T_3 \equiv \bigwedge_{\substack{u \in T_1 \\ v \in T_2 \\ w \in T_3}} (u \vee v \vee w), \tag{3.3}$$

for every assignment of $x_1, \ldots, x_n$, both sides evaluate to the same boolean value. Notice that the length of the 3-CNF can be much bigger (but still polynomial): 3-DNF is at most as $3n$ but 3-CNF could be up to n^3. In a sense, this corresponds to feature generation.

Remark 3.4. Notice that the reverse is NOT true. Given a 3-CNF it might not be representable as a 3-DNF.

Another important point to notice is that after the transformation into 3-CNF we are changing the distribution of the initial input space. But the definition of PAC learnable allows for *any* distribution, so we are fine.

The upshot here is that we can learn 3-CNF by 3-CNF but this problem contains the intractable problem of learning 3-DNF by 3-DNF. The trick is that we have a bigger solution space so we the 3-CNF algorithm is fed a 3-DNF, it has the option to output something outside the 3-DNF class, namely a 3-CNF.

Theorem 3.5. *3-CNF is PAC-learnable and 3-DNF. Further, by the discussion above, 3-DNF is learnable "by" 3-CNF.*

This motivates the more general definition for PAC-learnable that takes into account the solution class.

Definition 3.6 (PAC Learning)**.** A class $\mathcal{C}$ is *Probably Approximately Correct (PAC) learnable* by $\mathcal{C} \subset \mathcal{H}$ if there exists an algorithm, L, such that:

$$\forall c \in \mathcal{C}, \quad \forall D \text{ over } X, \quad \forall \varepsilon, \delta > 0 \tag{3.4}$$

- (Learning) With probability $\geq 1-\delta$, L outputs a hypothesis, $\underline{h \in \mathcal{H}}$ such that $D[h \triangle c] < \varepsilon$, i.e. we have error at most ε

$$Err(h) \coloneqq \mathbb{P}_{x \sim D}[h(x) \neq c(x)] \leq \varepsilon. \tag{3.5}$$

- (Efficient) L runs in time/sample *poly* $\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n\right)$. *Samples & computation.*

Remark 3.7. $\mathcal{C}$ is usually called the target class and $\mathcal{H}$ the hypothesis class.

3.2 Consistency implies PAC-learnable

- Suppose we have some target and hypothesis classes, $\mathcal{C} \subset \mathcal{H}$.
- Let A be a *consistent* algorithm:
  - i) Given $\underline{\text{any}}$ finite sample, $S = \{\langle x_1, y_1 \rangle, \ldots, \langle x_m, y_m \rangle\}$, where for all i we have $y_i = c(x_i)$ for some $c \in \mathcal{C}$.

ii) A outputs $h \in \mathcal{H}$ such that $h(x_i) = y_i = c(x_i)$ for all i.

Theorem 3.8. *For any finite $\mathcal{H}$, a consistent algorithm for $\mathcal{H}$ is a PAC-learnable algorithm.*

Proof. We call a hypothesis, $h \in \mathcal{H}$, ε-bad if $Err(h) > \varepsilon$. Now we generate a size-m S according to $Ex(c, D)$, which is the subroutine that generates samples from D. For any fixed ε-bad h, we have an upper bound on the probability of h being consistent with S

$$\mathbb{P}_S[h \text{ consistent with } S] \leq (1-\varepsilon)^m. \tag{3.6}$$

Therefore, $\mathbb{P}_S[\exists h \in \mathcal{H}$ that is both ε-bad and consistent with $S] \leq |\mathcal{H}|\,(1-\varepsilon)^m$. Now we choose $\delta > 0$ such that $|\mathcal{H}|\,(1-\varepsilon)^m < \delta$. We can conclude that as long as $m \geq \frac{const}{\varepsilon} \ln \frac{|\mathcal{H}|}{\delta}$ the PAC learning definition will be satisfied. □

Remark 3.9. $|\mathcal{H}|\,(1-\varepsilon)^m \leq e^{\ln|\mathcal{H}| + m\ln(1-\varepsilon)} \approx e^{\ln|\mathcal{H}| - c_0\varepsilon m} \to 0$ as $m \to \infty$. A key point of this is that we can let the complexity of the hypothesis to grow as the sample size gets bigger and keeping this quantity under control. That is, the more data you have, the more complex the model you can train without over-fitting too much. If $\mathcal{H}_m$ is the hypothesis for data of size m, we only need $\ln|\mathcal{H}_m| \leq c \cdot m^\beta$, for some $\beta < 1$ and $c = c(dim)$. From here we obtain $m \geq \left(\frac{c}{\varepsilon}\right)^{\frac{1}{1-\beta}}$.

Next we want to deal with the case of a possibly infinite hypothesis class $\mathcal{H}$. The first approach could be to try and force feed a discretization of the hypothesis into the finite case one but this might not work because it depends on the interaction between the distribution of the data and the chosen discretization. We want something better.

- Class $\mathcal{H}$ over X.
- $h \in \mathcal{H}$ as functions $h(x) \in \{0,1\}$; or as sets $h \subset X$.
- Let $S = \{x_1, \ldots, x_m\}$ be an ordered subset of X.
- $\Pi_{\mathcal{H}}(S) = \big\{\langle h(x_1), \ldots, h(x_n)\rangle : h \in \mathcal{H}\big\} \subseteq \{0,1\}^m, \quad \big|\Pi_{\mathcal{H}}(S)\big| \leq 2^m.$

  Remark 3.10. If the inclusion above is saturated then it means that our hypothesis can classify ANY labeling of that size. That is bad for learning because it is saying that there is no structure in the data.

  Remark 3.11. For each variable in S, say x_i, it induces a partition on the class $\mathcal{H}$ according on what is the value of $h(x_i)$ for $h \in \mathcal{H}$. Then, one can think about $\Pi_{\mathcal{H}}(S)$ as the collection of all the the possible labelings with concepts in the class $\mathcal{H}$.

Definition 3.12. We say that S *is shattered* if the the equality case holds, $\Pi_{\mathcal{H}}(S) = \{0,1\}^m$, or equivalently, $\big|\Pi_{\mathcal{H}}(S)\big| = 2^m$.

Definition 3.13. The *Vapnik-Chervonenkis dimension of* $\mathcal{H}$ as

$$VCD(\mathcal{H}) = \text{ size of the largest shattered set} \tag{3.7}$$

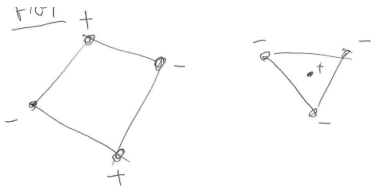
$$= \max_d \Big\{\exists S \subseteq X : |S| = d \quad \& \quad \Pi_{\mathcal{H}}(S) = \{0,1\}^d\Big\}. \tag{3.8}$$

Example 3.14. If $\mathcal{H}$ is finite, then $VCD(\mathcal{C}) \leq \ln|\mathcal{C}|$ because shattered d points requires 2^d functions. Equivalently, if $VCD(\mathcal{C}) = d$, then we must have $|\mathcal{C}| \geq 2^d$.

Example 3.15 (Rectangles in $\mathbb{R}^2$)**.** The VCD dimension is 4 in this case because the rectangular convex hull of a set of points is always given by the four extremal points in each direction. So there are always impossible labelings with 5 points. In general, the same kind of argument shows that the VCD dimension is $2d$.

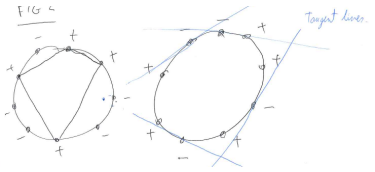
4 Lecture 4: 2017.02.13

Example 4.1 (Hyperplanes in $\mathbb{R}^2$)**.** It is clear that the VCD is at least 3 because given three points, we can always choose a line that separates the points in any way we want. In fact, the VCD is exactly 3.



In general, for hyperplanes in $\mathbb{R}^d$ the VCD dimension is $d + 1$.

Example 4.2 (Convex n-gons in $\mathbb{R}^2$)**.** In this case, $VCD = 2n + 1$. For example, if $n = 4$, $VCD = 9$.



Remark 4.3. Note that in these geometric situations, the VCD dimension is linearly related to the free parameters in our model. This is generally true (more or less). This supports the idea that we need more samples than parameters to be able to start learning something significant.

Remark 4.4 (Monotone conjunctions over $\{0,1\}^m$ e.g. $x_1 \wedge x_5 \wedge x_7$). We claim that $VCD \geq n$. Indeed, consider the bit vectors: $\langle 01\ldots1\rangle, \langle 101\ldots1\rangle, \ldots, \langle 1\ldots10\rangle$. We can always find a conjunction with the desired label. Namely, $x_1 \wedge \ldots \wedge x_i^{\vee} \wedge \ldots \wedge x_n$ or its negation as needed.

Overloaded Notation

$$\Pi_{\mathcal{C}}(m) \coloneqq \max_{S\,:|S|=m} \left\{\left|\Pi_{\mathcal{C}}(S)\right|\right\} \tag{4.1}$$

$$= \text{max. \# of labelings induced by } \mathcal{C} \text{ on m points.} \tag{4.2}$$

Note that we have the following:

- $\Pi_{\mathcal{C}}(m) \leq 2^m$.
- If $m \leq VCD(\mathcal{C})$, $\Pi_{\mathcal{C}}(m) = 2^m$.
- If $m > VCD(\mathcal{C})$, $\Pi_{\mathcal{C}}(m) < 2^m$.

Theorem 4.5. *Let $VCD(\mathcal{C}) = d$. Then $\Pi_{\mathcal{C}}(m) = O(m^d)$.*

Remark 4.6. As the sample size becomes large compared to the dimension, the growth rate of $\Pi_{\mathcal{C}}(m)$ is sub-exponential. In fact, the fraction of possible behaviors grows like $\frac{\frac{1}{m^d}}{2^m}$ which tends to 0 as $m \to \infty$.

Proof of Theorem 4.5

Consider $\phi_d(m) \coloneqq \phi_d(m-1) + \phi_{d-1}(m-1)$ with boundary conditions $\phi_0(m) = \phi_d(0) = 1$. We first prove the following lemma.

Lemma 4.7. *If $VCD(\mathcal{C}) = d$, then for any m we have $\Pi_{\mathcal{C}}(m) \leq \phi_d(m)$.*

Proof. We use a double induction argument. The base cases are clear so assume the lemma is true for any $d' \leq d$ and $m' \leq m$ with at least one the inequalities being $<$. Consider the set $S = \{x_1, \ldots, x_m\}$. Recall that $\Pi_{\mathcal{C}}(S)$ is just the possible labelings of S that can be realized with concepts in $\mathcal{C}$. Now consider all the possible labelings that are possible for $\{x_1, \ldots, x_{m-1}\} = S \setminus \{x_m\}$. For each of these, there are exactly 2 potential labelings for S depending on whether $x_m = 0$ or $x_m = 1$. Then, by the induction hypothesis, we have

$$\left|\Pi_{\mathcal{C}}(S \setminus \{x_m\})\right| \leq \Pi_{\mathcal{C}}(m-1) \leq \phi_d(m-1). \tag{4.3}$$

Next consider the labelings of $S \setminus \{x_m\}$ that appear twice in the labelings of S. That is, those such that both $x_m = 0$ and $x_m = 1$ are possible labelings. Then we can shatter at most $d-1$ points with $S \setminus \{x_m\}$ because otherwise we would violate $VCD(\mathcal{C}) = d$. Finally, by induction hypothesis we have that

$$\#\text{ of labelings of } S \setminus \{x_m\} \ \leq \phi_{d-1}(m-1). \tag{4.4}$$

Combining these two bound we obtain the result. □

Claim 4.8. $\phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$.

Proof. By induction,

$$\phi_d(m) = \phi_d(m-1) + \phi_{d-1}(m-1) = \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} \tag{4.5}$$

$$= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d} \binom{m-1}{i-1} \tag{4.6}$$

$$= \sum_{i=0}^{d} \left[\binom{m-1}{i} + \binom{m-1}{i} \right] = \sum_{i=0}^{d} \binom{m}{i}. \tag{4.7}$$

□

Now the proof of Theorem 4.5 follows by noting that $\sum_{i=0}^{d} \binom{m}{i} = O(m^d)$.

4.1 Combinatorial → Probabilistic

Fix target $c \in \mathcal{C}$ and a distribution D. Thinking of c as a set, we can consider $\mathcal{H}\triangle c = \{h\triangle c : h \in \mathcal{H}\}$, which are the *error regions*.

Definition 4.9. We say that S of m points is a *ε-net* for $\mathcal{H}\triangle c$ if for all $r \in \mathcal{H}\triangle c$ such that $D[r] \geq \varepsilon$, then we have $S \cap r \neq \emptyset$.

Remark 4.10. $VCD(\mathcal{H}\triangle c) = VCD(\mathcal{H})$. This is because taking the symmetric difference corresponds to a XOR operation.

Consider the auxiliary function on $S = \{x_1, \ldots, x_m\}$

$$A(S) \coloneqq \begin{cases} 1 & \text{if } S \text{ is not an } \varepsilon\text{-net for } \mathcal{H}\triangle c \\ 0 & \text{else} \end{cases}, \tag{4.8}$$

$$B\left(S, S'\right) \coloneqq \begin{cases} 1 & \text{if } \exists r \in \mathcal{H}\triangle c \text{ st. } D[r] \geq \varepsilon,\ r \cap S = \emptyset, |r \cap S'| \geq \frac{\varepsilon m}{2} \\ 0 & \text{else} \end{cases}. \tag{4.9}$$

Remark 4.11. Event $B\left(S, S'\right)$ occurs if $\exists r \in \mathcal{H}\triangle c$ st. r is "hit":

- $l \geq \frac{\varepsilon m}{2}$ times in $S \cup S'$ but *all* l fall in S'.

$$\mathbb{P}_{S,S'}[B(S,S')] = \mathbb{P}_{S,S'}[B(S,S') \mid A(S)] \cdot \mathbb{P}_{S,S'}[A(S)] \tag{4.10}$$

$$\geq \frac{1}{2}\mathbb{P}_{S,S'}[A(S)], \tag{4.11}$$

where we used Chebyshev's inequality on the last line.

Draw $\langle S, S'\rangle$ of $2m$ points in 2 steps:

1. Draw $2m$ points randomly from the distribution D. $x_1, \ldots, x_m, x_{m+1}, \ldots, x_{2m}$
   - # possible labelings of the $2m$ points by $\mathcal{H}\triangle c$ fixed $\leq \phi_d(2m)$.
2. Split the $2m$ points randomly into S and S'.

Claim 4.12. $\mathbb{P}_{permutation}[B\left(S, S'\right)] = \frac{\binom{m}{l}}{\binom{2m}{l}} \leq 2^{-l}$.

Proof. Simple exercise in combinatorics. □

Then we have that

$$\mathbb{P}[B\left(S, S'\right)] \leq \phi_d(2m)2^{-\frac{\varepsilon m}{2}} \leq C_0(2m)^d 2^{-\frac{\varepsilon m}{2}} \leq e^{C_1[d\log(m) - \varepsilon m]}. \tag{4.12}$$

We have just proven the following theorem.

Theorem 4.13. *As long as $m \geq C_2\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right) + \frac{d}{\varepsilon}\log\left(\frac{1}{\varepsilon}\right)\right)$, where $VCD(\mathcal{H}) = d$, we have that with probability $\geq 1 - \delta$, any* consistent *$h \in \mathcal{H}$ has error $\leq \varepsilon$ with respect to c and D.*

5 Lecture 5: 2017.02.20

Outline for today:

1. VC theory rcap
2. Matching lower bound
3. VC++:
   - unrealizable/agnostic case
   - ERM/SRM
   - Rademacher complexity, sparsity, etc.
   - general loss functions
4. Learning with noise

5.1 VC theory review

Recall from last time that we defined $\Pi_{\mathcal{C}}(m)$ as the max. # of labelings induced by $\mathcal{C}$ on m points, by definition we have the naive bound $\Pi_{\mathcal{C}}(m) \leq 2^m$. One of the highlights is that we were able to prove that

$$\Pi_{\mathcal{C}}(m) \leq \phi_d(m) \leq O(m^d). \tag{5.1}$$

This bound is saying that past the VCD dimension, the number of possible labelings we can achieve is only a increasingly small fraction of the total number (2^m). This was the key step to prove the following theorem

Theorem 5.1. *Given $c \in \mathcal{C}$ and a distribution, D, over X. If we take $m \geq c_0 \frac{d}{\varepsilon} \ln \frac{1}{\delta}$ examples, where $VCD(\mathcal{C}) = d$, then, with probability $\geq 1 - \delta$, any $h \in \mathcal{C}$ that is consistent with the sample has $Err(h) \leq \varepsilon$.*

Remark 5.2. This is a purely "information theory" statement, we are ignoring how hard is to actually find an appropriate h.

5.2 Matching lower bound

The converse of the theorem above is not true, in other words, we can find distributions such that is not necessary to take that many examples. For example, consider the case of the triangles in dimension 2 and a distribution that has support on a single point. This motivates the next result.

Theorem 5.3. *Given $\mathcal{C}$ with $VCD(\mathcal{C}) = d$, there exists D a distribution over X such that $\Omega(d/\varepsilon)$ examples are required to learn with error $\leq \varepsilon$.*

Proof. Consider $\{x_1, \ldots, x_d\}$ a shattered set and let D be the uniform distribution over this set. Now, since the set is shattered, we can choose a function $c_i \in \mathcal{C}$ with $i = 1, \ldots, 2^d$ for each of the possible labeling of these d points. Now we randomly choose the target concept, c, among these c_i. This is equivalent to flipping a fair coin d times to determine the labeling induced by c on S.

Now we let L be any PAC learning algorithm with the data above. Given error parameter $\varepsilon \leq \frac{1}{8}$ suppose we only draw $m < d$ examples. Say that the number of distinct points that we have seen is $m' \leq m$, for the remaining of the points the problem is equivalent to predicting a fair coin toss. Therefore, the expected error on the whole sample is $\frac{\frac{d-m'}{2}}{d} = \frac{d-m'}{2d}$. By Chebyshev's inequality,

$$\mathbb{P}\left(Error \geq \frac{d-m'}{4d}\right) \leq \frac{1}{2}. \tag{5.2}$$

In particular, if $m = \frac{d}{2}$, the error is at least $\frac{1}{8}$ with probability at least $\frac{1}{2}$ and the algorithm fails the conclusion of the theorem when c is chosen randomly. This implies that there exist some target c on which L fails, so we need at least $\Omega(d)$ sample complexity lower bound.

Finally, to incorporate the ε onto the lower bound, we need only scale the distribution above. Modify D so that the point x_1 has probability $1-8\varepsilon$ and let the rest have probability $\frac{8\varepsilon}{d-1}$. This results in needing to draw more points to be in the same set up as before. Details are left as exercise or can be checked on the textbook in page 63. □

5.3 VC++

5.3.1 Unrealizable/Agnostic setting

So far we have assumed that the the target $c \in \mathcal{C}$. That is, that the truth is perfectly represented by our chosen class. Now we drop this assumption. We have a *hypothesis* class $\mathcal{H}$ but $c \notin \mathcal{H}$, i.e. the target might not be representable by $\mathcal{H}$. The first thing we need to lower our expectations for learning.

Definition 5.4. $h^* := \underset{h \in \mathcal{H}}{argmin} \{Err(h)\}$, where $Err(h) = \mathbb{P}_{X \sim D}[h(x) \neq c(x)]$. We also define $\varepsilon^* = Err(h^8) =$ best possible error in $\mathcal{H}$.

Let's set up some additional notation. Given $h \in \mathcal{H}$ and a labeled sample

$$S = \{\langle x_1, c(x_1)\rangle, \ldots, \langle x_m, c(x_m)\rangle\}, \tag{5.3}$$

we have some different types of errors:

- $Err(h)$, the *true error* as aboe
- $\widehat{Err}(h) = \frac{1}{m}\left|\{\langle x_i, y_i\rangle \in S \ : \ h(x_i) \neq y_i\}\right|$, the *empirical error.*

Theorem 5.5. *Given any target c (not necessarily in $\mathcal{H}$) and a distribution D over X, if we take $m \geq c_0 \frac{d}{\varepsilon^2} \ln \frac{1}{\delta}$ where $VCD(\mathcal{H}) = d$, then with probability $\geq 1-\delta$, for all $h \in \mathcal{H}$ we have* uniform *convergence*

$$\left|\widehat{Err}(h) - Err(h)\right| \leq \varepsilon. \tag{5.4}$$

Proof. According to Prof. Kearns is a modification of the analogous result where $c \in \mathcal{H}$. He mentioned that some extra materials might be uploaded to the course website. □

Remark 5.6. To apply this theorem, in applications, we find/compute $\hat{h}^* = \underset{h \in \mathcal{H}}{argmin} \left\{ \widehat{Err}(h) \right\}$. By the theorem, we have $\left| \widehat{Err}(\hat{h}^*) - Err(\hat{h}^*) \right| \leq \varepsilon$ and $\left| \widehat{Err}(h^*) - Err(h^*) \right| \leq \varepsilon$. This implies that,

$$\left| Err(\hat{h}^*) - Err(h^*) \right| \leq 3\varepsilon. \tag{5.5}$$

5.3.2 Structural Risk Minimization

Fix some target c, a distribution D and a sample of size m S.Suppose we have $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_d \subset \ldots$. For example, each hypothesis could be a neural network with an increasing number of hidden layers. Note that the VCD dimension can only increase along the nested sequence. For simplicity, let's assume that $VCD(\mathcal{H}_d) = d$. Define

$$\begin{aligned} Err(d) &\coloneqq \min_{h \in \mathcal{H}_d} \{Err(h)\} && (5.6) \\ &= \text{ best true error in } \mathcal{H}_d. && (5.7) \\ \widehat{Err}(d) &\coloneqq \min_{h \in \mathcal{H}_d} \{\widehat{Err}(h)\} && (5.8) \\ &= \text{ best empirica error on } S \text{ in } \mathcal{H}_d. && (5.9) \end{aligned}$$

Remark 5.7. From the theorem above, $|Err(d) - \widehat{Err}(d)| \lesssim \sqrt{\frac{d}{m}}$

If we use minimization of the empirical error as the criterion to choose our model, we might run into trouble in the form of overfitting. Basically, the empirical error can only decrease as the complexity of the model increases but it can deviate from the true error. Luckily for us, the theorem above gives a way to correct for this by instead finding

$$\min_d \left\{ \widehat{Err}(d) + \sqrt{\frac{d}{m}} \right\}. \tag{5.10}$$

5.3.3 Refinements

In the theorem from last time, $\Pi_{\mathcal{C}}(S) \to \Pi_{\mathcal{C}}(m) \leq O(m^d)$. But if one is more careful in the proof, it is possible to arrive to a bound on $\mathbb{E}_{S \sim D,c}[\ln \left| \Pi_{\mathcal{C}}(S) \right|]$ where $|S| = m$, the *VC entropy*. (cf. Book)

5.3.4 General loss functions

The assumptions are as follows:

- observations $z \sim D$ iid. E.g. $z = \langle x, y \rangle$ with $y \in \{0, 1\}$; $z = \langle x, y \rangle$ with $y \in \mathbb{R}$; $z = x$.
- models $h \in \mathcal{H}$.

- $\mathbb{R}$-valued loss function, $L(h, z)$. E.g.

  1. Classification: $L(h, \langle x, y\rangle) = 1$ if $h(x) \neq y$, or 0 otherwise.
  2. Linear regression: $L(h, \langle x, y\rangle) = (h(x) - y)^2$
  3. $L(h, x) = \ln \frac{1}{h(x)}$

  We'd like to minimize $\mathbb{E}_{z\sim D}[L(h, z)]$

Remark 5.8. Although we are not going to touch on that, there is a theorem like the one we proved for this setting as well.

5.3.5 Analog to VCD

Take d observations $z_1, \ldots, z_d$ and consider the following set $T = \{\langle L(h, z_1), \ldots, L(h, z_d)\rangle : h \in \mathcal{H}\}$. The analog to the VC dimension would be to look at this cloud of points a see if it is "space-filling" or that not all points lie on some lower dimensional subspace. The criterion is that you want all the d-dimensional orthants to be intersected by T.

6 Lecture 6: 2017.02.27

6.1 Learning with Noise

Now we turn our attention to learning with noise. The set up is the same as in the PAC-learning with the difference that the "subroutine" that extracts examples, $EX_\eta(c, D)$, is now a Bernoulli trial. With probability $1 - \eta$ it returns the correct label $\langle x, c(x)\rangle$ and with probability η it returns the incorrect label $\langle x, \neg c(x)\rangle$. We also ask for the algorithm to be polynomial on $\frac{1}{1-2\eta}$.

- We assume that the noise is *independent* of x and $c(x)$. This is akin to measurement error. Note that our noise affects only the labels and not on the x's.
- We also assume that $\eta < \frac{1}{2}$. Otherwise it would be impossible to distinguish between the true label and the false label.

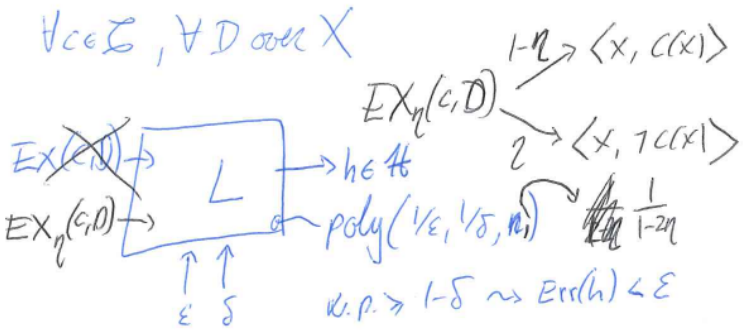


Figure 2: PAC-learning with noise.

Given any $h \in \mathcal{H}$, we can compute the noisy error

$$\mathbb{P}_{\langle x,y\rangle \sim EX_\eta}[h(x) \neq y] = (1-\eta)Err(h) + \eta(1 - Err(h)) = \eta + (1-2\eta)Err(h), \tag{6.1}$$

observe that since we have $\eta < \frac{1}{2}$, this is increasing as a function $Err(h)$. This means that noise preserves the ranking of hypothesis according to their error. So we can still minimize this quantity and obtain a good model.

Remark 6.1. As a consequence, all the VCD results still hold with the only caveat that we need more data, in particular, we need

$$m \sim \frac{VCD(\mathcal{C})}{\varepsilon^2(1-2\eta)^2} \log\left(\frac{1}{\delta}\right). \tag{6.2}$$

6.1.1 "Malicious" Errors

Same setup as before but with a slight change.

- with probability $1-\eta$ we return the correct pair $\langle x, c(x)\rangle$,
- with probability η, an "adversary" generates *any* pair $\langle x, y\rangle$.

The general theory still applies and by minimizing observed error we can still pick up a target with error at most η. However, the "malicious" problem is considerably worse and we can no longer ask for $Err << \eta$. To see this consider a fixed distribution D over X and two concepts $c_1, c_2 \in \mathcal{C}$ such that $c_1 \triangle c_2$ has ε weight under D.

Figure 3: Malicious noise.

Then, as long as $\eta \geq \frac{\varepsilon}{1+\varepsilon}$, the adversary can make you confuse c_1 and c_2.

6.2 Learning (monotone) Conjunctions with "Statistics"

Recall the problem of learning conjunctions.

- $X = \{0,1\}^n$, concept class. E.g. $c = x_1 \wedge x_5 \wedge x_6 \wedge x_{19}$.

We started with the hypothesis $h = x_1 \wedge \ldots \wedge x_n$ and we deleted any variable that contradicted the positive examples (all the variables that are zero in some positive example). However, if we introduce noise this algorithm falls apart since we could have the all 0's vector to be incorrectly labeled positive which results in the empty conjunction. The underlying problem is that we made drastic decisions on our hypothesis based on a single examples. To overcome this shortcoming, we will only introduce changes if we see enough *statistical* evidence.

- For each x_i, define

$$p_0(x_i) = \mathbb{P}_{\vec{x}\sim D}[x_i = 0], \tag{6.3}$$
$$p_{01}(x_i) = \mathbb{P}_{\vec{x}\sim D}[x_i = 0 \;\&\; c(\vec{x}) = 1]. \tag{6.4}$$

  We call x_i *significant* if $p_0(x_i) \geq \frac{\varepsilon}{4n}$, and *harmful* if $p_{01}(x_i) \geq \frac{\varepsilon}{4n}$.

- Algorithm: Let h be the conjunction of all x_i's that are significant and not harmful.

Let's look at the analysis of the errors. For the FP type

$$\mathbb{P}_{\vec{x}\sim D}[c(\vec{x}) = 0 \;\&\; h(\vec{x}) = 1], \tag{6.5}$$

must be some x_i such that $x_i \in c$, $c_i \notin h$ and $x_i = 0$. This means that x_i is not harmful which implies that x_i is not significant and $p_0(x_i) \leq \frac{\varepsilon}{4n}$. Thus, the total error is $p_0(n) \leq n\frac{\varepsilon}{4n} \leq \frac{\varepsilon}{4}$. Similarly, for the FN type

$$\mathbb{P}_{\vec{x}\sim D}[c(\vec{x}) = 1 \;\&\; h(\vec{x}) = 0], \tag{6.6}$$

must be some x_i such that $x_i \notin c$, $x_i \in h$ and $x_i = 0$. Therefore, $p_{01}(n) \leq \frac{\varepsilon}{4n} n \leq \frac{\varepsilon}{4}$.

Remark 6.2.

Remark 6.3 (Concentration Inequalities). Consider a biased coin with $\mathbb{P}[head] = p$ and $\mathbb{P}[tails] = 1 - p$. We flip the coin m times and let $\hat{p} = \frac{\#\text{ heads observed}}{m}$. We are interested in bounds like (Hoeffding's)

$$\mathbb{P}_{\text{m flips}}\left[|p - \hat{p}| \geq \varepsilon\right] \leq e^{-\frac{\varepsilon^2 m}{3}}, \tag{6.7}$$

and the closely related Chernoff bound. This is like the law of large numbers but with an explicit rate.

We can estimate the $p_0(x_i)$ probabilities with $EX_\eta(c, D)$ since we don't care about the noise of the labels.

The interesting part is that we can do the same for the p_{01}'s.

6.3 Statistical Query (SQ) Learning

Remark 6.4 (Warning). The professor sort of rushed through the definition of SQ-learning. Check the textbook for more details.

We modify the PAC learning algorithm. We replace the subroutine $EX(c, D)$ with $SQ(c, D)$ which interacts with the algorithm L in the role of "oracle", answering the questions (queries) posed by L. For example, what is the probability of $x_i = 0$?.

These queries, χ, are *predicates* (0/1 valued functions) on $\langle x, y\rangle$ pairs

$$\chi(x, y) \in \{0, 1\}^n \longrightarrow P_\chi \coloneqq \mathbb{P}_{\langle x,y\rangle \sim EX(c,D)}[\chi(x, y) = 1]. \tag{6.8}$$

E.g. $\chi(\vec{x}, y) = 1 \iff x_{11} = 0$ $[p_0(x_{11})]$
E.g. $\chi(\vec{x}, y) = 1 \iff x_{11} = 0 \ \& \ y = 1$ $[p_{01}(x_{11})]$

On query χ, $SQ(c, D)$ returns any value $\hat{P}_\chi \in \left[P_\chi - \tau, P_\chi + \tau\right]$. To say that $\mathcal{C}$ is *SQ-learnable* if we require $\tau \geq poly\left(\frac{\varepsilon}{n}\right)$

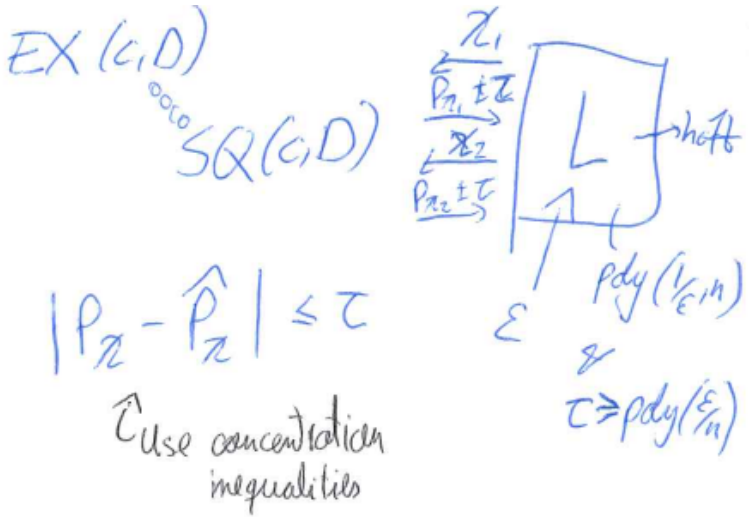


Figure 4: Statistical Query Learning.

Theorem 6.5 (uninteresting)**.** *If $\mathcal{C}$ is SQ-learnable, then $\mathcal{C}$ is PAC-learnable.*

Theorem 6.6. *If $\mathcal{C}$ is SQ-learnable, then $\mathcal{C}$ is PAC-learnable <u>with</u> noise.*

Proof. We want to estimate

$$P_\chi \coloneqq \mathbb{P}_{\langle x,y\rangle \sim EX(c,D)}[\chi(x, c(x)) = 1], \tag{6.9}$$
$$P_\chi^\eta \coloneqq \mathbb{P}_{\langle x,y\rangle \sim EX^\eta(c,D)}[\chi(x, y) = 1], \tag{6.10}$$

but we don't have access to the $EX(c, D)$ subroutine. Consider

$$X_2 \coloneqq \{x \in X \ : \ \chi(x, 0) = \chi(x, 1)\}, \tag{6.11}$$

and define the following partition of X

$$X_1 \coloneqq \{x \in X \ : \ \chi(x, 0) \neq \chi(x, 1)\}, \tag{6.12}$$

X_1 and X_2 are disjoint and clearly $X_1 \cup X_2 = X$. Now consider $p_1 = D[X_1]$ and $p_2 = 1 - p_1 = D[X_2]$. For $x \in X$ we have the normalized conditioned distributions

$$D_1[X] = \frac{D[X]}{p_1} \to P_\chi^1 = \mathbb{P}_{EX(c,D_1)}[\chi = 1], \tag{6.13}$$

$$D_2[X] = \frac{D[X]}{p_2} \to P_\chi^2 = \mathbb{P}_{EX(c,D_2)}[\chi = 1]. \tag{6.14}$$

This just introduced notation. By considering conditioned probabilities, we now can compute

$$P_\chi^\eta = (1-\eta)P_\chi + \eta \left(p_1 \underbrace{\mathbb{P}_{x\sim D_1, y=\neg c(x)}[\chi(x,y)=1]}_{\mathbb{P}_{EX(c,D)}[\chi(x,c(x))=0]=:1-P_\chi^1} + p_2 \underbrace{\mathbb{P}_{x\sim D_2, y=\neg c(x)}[\chi(x,y)=1]}_{\mathbb{P}_{EX(c,D_2)}[\chi(x,c(x))=1]=:P_\chi^2} \right) \tag{6.15}$$

$$= (1-\eta)P_\chi + \eta \left(p_1(1-P_\chi^1) + p_2 P_\chi^2 \right). \tag{6.16}$$

We can solve this for P_χ

$$P_\chi = \frac{1}{1-\eta} \left[P_\chi^\eta - \eta \left(p_1(1-P_\chi^1) + p_2 P_\chi^2 \right) \right]. \tag{6.17}$$

Bu now we can empirically estimate all the elements in the RHS using noisy data. The only tricky one is P_χ^1 but we have

$$P_\chi^1 = \frac{P_\chi^\eta - p_2 P_\chi^2 - \eta p_1}{(1-2\eta)p_1}, \tag{6.18}$$

so we are fine. To finish the proof in detail one has to do a robustness analysis of the expression that we are using to estimate P_χ. See the textbook for more details. □

Claim 6.7. *"Almost" every PAC-learning algorithm has an SQ-algorithm.*

7 Lecture 7: 2017.03.13

Outline

- SQ model: review and lower bounds
- PAC learning and cryptography
- Boosting Part I

7.1 Query complexity in SQ

We saw that any PAC-learnable model can be framed in the SQ model by simulating the queries with enough draws from the distribution i.e calls to $EX(c, D)$. Intuitively, the query complexity of the obtained model should not violate the lower bounds from previous lectures on the VCD.

Claim 7.1. *Assume $VCD(\mathcal{C}) = d$, then at least $\Omega\left(\frac{d}{\log d}\right)$ queries with $\tau = O(\varepsilon)$ are required in the SQ model.*

Sketch. Let $\{x_1, \ldots, x_d\}$ be shattered by $\mathcal{C}$. Let D such that x_1 has weight $1 - 3\varepsilon$ and the rest have uniform weight $\frac{3d}{d-1}$. Consider the query $\chi = 1 \iff x = x_1 \;\&\; y = 0$, this allows the algorithm to learn the label of x_1. Now let $b_i = c(x_i)$, we must learn the labels of the majority of these labels. Specifying some other labels c_i, we can make statistical queries like $\mathbb{P}_i[b_i = c_i]$. For example, "what is the proportion of 1's among b_i's?". This statistical query can be thought of as the inner product of this unknown $\vec{b}$ and any $\vec{c}$ that we want. Therefore, the learning problem reduces to a linear algebra problem. □

Recall the following remark from last time.

Claim 7.2. *"Almost" every PAC-learning algorithm has an SQ-algorithm.*

We can actually find counterexamples to the claim, hence the "almost" in the statement.

7.1.1 PAC-learnable but not SQ-learnable

Let $X = \{0, 1\}$ and consider the parity functions over X. Namely, for any subset $S \subset \{1, \ldots, n\}$ we can define

$$f_S(\vec{x}) = \sum_i x_i \mod 2 \in \{0, 1\}, \tag{7.1}$$

and the associated pairs $\langle \vec{x}, f_S(\vec{x}) \rangle$. They count the parity of the 1 bits in the set S. Note that $|\mathcal{C}| = 2^n$. It turns out that this is PAC-learnable because examples correspond to linear equations but you can't can get the same information statistically. We have the info-theoretical result.

Theorem 7.3. *Parity functions satisfy the following:*

i) PAC-learnable (poly time & samples)

ii) Provably require exponential in n queries in SQ.

Intuition. Let D be uniform over X.

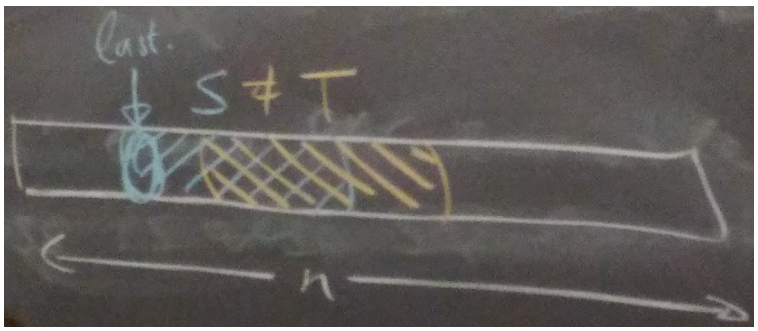


Figure 5: Parity functions.

If we flip coins to decide the bits, by the time we reach the last one, T is completely determined but S relies on the result completely.

$$\forall S \neq T, \quad \mathbb{P}[f_S(\vec{x}) = f_T(\vec{x})] = \frac{1}{2} = \mathbb{P}[f_S(\vec{x}) \neq f_T(\vec{x})] \tag{7.2}$$

Basically, the crux of the proof of ii) is proving that no matter what kind of queries we ask they reduce to the "Is the answer this function?" type of questions. Since, there are 2^n different functions we need an exponential amount of queries. □

We can rephrase parity functions in the following way. Take a target vector $\vec{c}$ in X,

$$f_S(\vec{x}) = \sum_i c_i x_i \mod 2, \tag{7.3}$$

then any pair $\langle \vec{x}, f_{\vec{c}}(\vec{x}) \rangle$ represents an algebraic equation on the unknowns c_i's. Therefore, this can be learned in polynomial time by a PAC-learning algorithm.

7.1.2 Characterizing Query Complexity in SQ

Definition 7.4. We define the SQ-dimension as

$$SQD(\mathcal{C}, D) \coloneqq \max_d \{\exists f_1, \ldots, f_d \in \mathcal{C} : \forall 1 \leq i \neq j \leq d, \\ \left|\mathbb{P}_D[f_i(x) = f_j(x)] - \mathbb{P}_D[f_i(x) \neq f_j(x)]\right| \leq \frac{1}{d^3}\Big\}. \tag{7.4}$$

Note that in the case of $\{0,1\}$, we can think of f_i with labels ± 1 as vectors with 2^n entries (corresponding to all subsets). Then, $\mathbb{P}_D[f_i(x) = f_j(x)] - \mathbb{P}_D[f_i(x) \neq f_j(x)]$ corresponds to the inner product of f_i and f_j. With this in mind, this definition is only saying that we want to find a large set of almost orthogonal vectors.

Theorem 7.5. *Let $SQ(\mathcal{C}, D) = d$, then at least $\Omega\left(d^{\frac{1}{3}}\right)$ queries with $\tau = O\left(\frac{1}{d^{\frac{1}{3}}}\right)$ are required to SQ-learn $\mathcal{C}$ wrt. D.*

Recall some of the PAC-learnable classes of functions we have seen.

- Conjunctions over $\{0,1\}^n$
- decisions lists (problem in the book)
- 3-term DNF by 3CNF.

The problem is that none of the are "universal" in the sense that they cannot represent all the boolean functions. In contrast,

general DNF functions: Expressions of the form $T_1 \vee \ldots \vee T_m$ where each T_i is a conjunction over $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$. Now, any boolean function can be represented as a look-up table. Since we can add a T_i for each of the entries of this table, they all can be represented by general DNF.

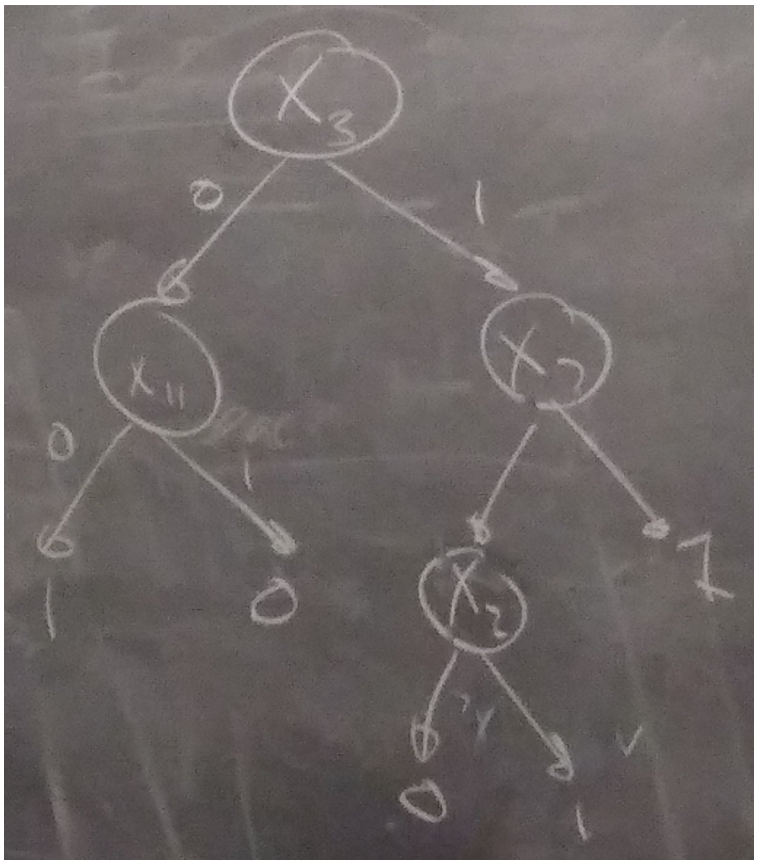


Figure 6: Decision tree.

decision trees: As before, we can copy the look-up table with a big enough table.

Remark 7.6. It is a big open problem whether these two classes are PAC-learnable. At least, we can use the theorem above to prove that they cannot be SQ learnable. This is in sharp contrast with the current state of the ML literature where most algorithms are SQ. This means that to PAC-learn these classes we would need radically different algorithms to the ones available at the moment.

Consider parity functions $f_S(\vec{x})$ where $|S| = \log(n)$. We have $\binom{n}{\log(n)} \approx n^{\log(n)}$. Then, $SQD = n^{\log(n)} \to \Omega(n^{\frac{\log(n)}{3}})$. Consider the parity function that looks at the first $\log(n)$ bits. We can write a decision-tree to compute the parity. This tree is a full binary tree of depth $\log(n)$.

7.2 Learning & Cryptography

Recall that we saw that PAC-learning 3-term DNF by 3-term DNF is hard assuming RP $\neq$ NP. We sidestepped this by showing that 3-term DNF is PAC-learnable by 3CNF. This raises the question

Q: Are there classes of functions that are hard to PAC-learn *no matter* what $\mathcal{H}$ is?

If we allow *any* hypothesis class we can always produce hypotheses that perform exhaustive search on the target class $\mathcal{C}$ until they find a consistent class with the sample. Then, everything is PAC learnable. We want to forbid these shenanigans.

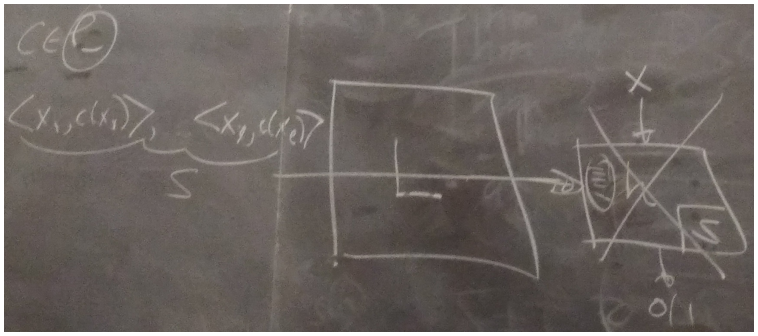


Figure 7: Overpowered hypotheses.

Q: Are there classes of functions that are hard to PAC-learn *no matter* what polynomially evaluable $\mathcal{H}$ we choose?

7.2.1 Cryptography Sidebar

- Alice wants to send Bob a message $\vec{x} \in \{0,1\}^n$ to Bob.
- Eve is listening to the conversation between them.
- Alice wants to encrypt her message.

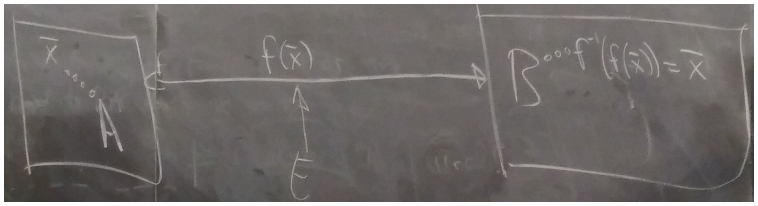


Figure 8: Cryptography layout.

One-time pad

- A & B meet privately and generate a long random bit string, $\vec{z} = z_1 \dots z_n$ where $z_i \in \{0,1\}$.
- Then A sends B $f(\vec{x}) = \vec{y}$ where $y_i = x_i \oplus z_i$.
- B can decrypt the message by doing another XOR with $\vec{z}$ since $(x_i \oplus z_i) \oplus z_i = x_i$
- The message appears indistinguishable from a random one. However, it can only be used once because Eve can take XOR with previous encryptions to find out the message.

- Not practical in the real world because you need to meet and agree on a $\vec{z}$ beforehand. This gives rise to public key cryptography.

Eve has a learning problem from previous deciphered messages.

Public Key Cryptography

- Replace shared key/secret with *two* keys for Bob:
  1. public encryption key, e
  2. private decryption key, d
- Desired data:
  - i) encryption $f_e(\vec{x})$ is easy to compute from e
  - ii) $g_d(f_e(\vec{x})) = \vec{x}$ easy to compute from d
  - iii) But $g_d(\vec{y})$ *hard* to compute from only e, even given many encrypted/decrypted pairs (bc anybody has the public keys and can generate examples of decrypted/encrypted messages)

Example 7.7 (RSA)**.** Bob chooses two random n-bit prime numbers, p and q and computes $N = pq$. Bob also generates a random n-bit string, e.

1. Bob's public key is (N, e)
2. To send a message x to Bob, you send $x^e \mod N$
3. From e, p and q Bob can compute a number d such that $(x^e \mod N)^d \mod N = x$. This easily satisfies i) and ii). Regarding iii), we only need that factorizing large integers is hard.

   Remark 7.8. For a slight modification of RSA, breaking it is equivalent to polynomial factorization algorithms.

Theorem 7.9 (Not exact statement)**.** *Under "standard assumptions" (i.e. factoring is hard (not in P), etc), then the following classes are not efficiently PAC-learnable by* any $\mathcal{H}$*:*

- *neural networks (complex enough)*
- *DFA*
- *boolean formuale*

which are universal representations.

Remark 7.10. Even if we only ask for $\varepsilon < \frac{1}{2}$, e.g. $\varepsilon = \frac{1}{2} - \frac{1}{poly(n, size(c))}$. It is also worth mentioning that these are worst-case results, this does not mean that the classes above are not useful in practice. This comes from the generality of the PAC learning paradigm where we require algorithms to work for any distribution D. Of course, one can always engineer distributions with better performance.

8 Lecture 8: 2017.03.20: Boosting

Recall the PAC-learning scheme:

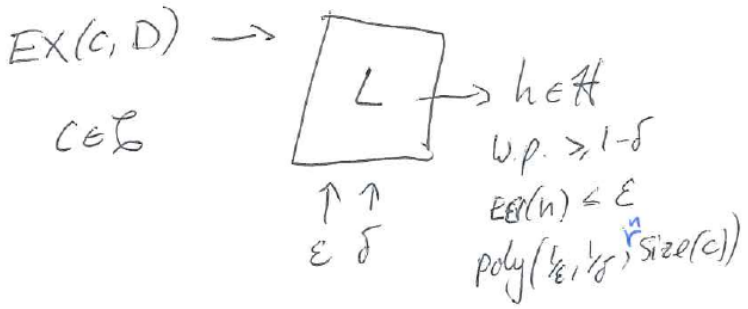


Figure 9: Scheme for PAC learning.

Hypothesis/Accuracy Boosting question:

If we have an algorithm for PAC-learning some class $\mathcal{C}$ but only for fixed ε_0, does this imply a full PAC algorithm (i.e. $\varepsilon \to 0$)

Remark 8.1. Today we see that the answer to this question is Yes. By the remark above, this implies that the proof must use the fact that we are allowing *any* distribution.

8.0.1 Confidence Boosting

- Given algorithm L that can achieve error ε ($\varepsilon \to 0$), but only for some *fixed* value of $\delta = \delta_0$ (e.g. $\delta_0 = \frac{9}{10}$) w.p. $1 - \frac{9}{10} = \frac{1}{10}$
- Run L k times, we get k hypothesis $h_1, \ldots, h_k$, which are independent. Note that each hypothesis has a chance $\frac{1}{10}$ of having error less than ε_0 so if k is big enough, choosing the one with the best error has as high probability as we want to achieve error less than ε_0. (It is basically the minimum of k independent Bernoulli trials).

$$\mathbb{P}_{S_1,\ldots,S_k \sim D}\left[\forall 1 \le i \le k, \quad Err(h_i) > \varepsilon\right] \le \delta_0^k, \tag{8.1}$$

  which is less than any chosen δ if k is big enough. It is enough to pick $k \ge \frac{\log\left(\frac{1}{\delta}\right)}{\log\left(\frac{1}{\delta_0}\right)}$

8.0.2 Accuracy Boosting

Given algorithm L such that for any distribution, w.p $\ge 1-\delta$, outputs h such that $Err(h) \le \varepsilon_0 = \frac{1}{2} - \gamma$ (i.e. slightly better than random guessing) (weak learning algorithm). One my try a similar thing as above but with taking a majority vote:

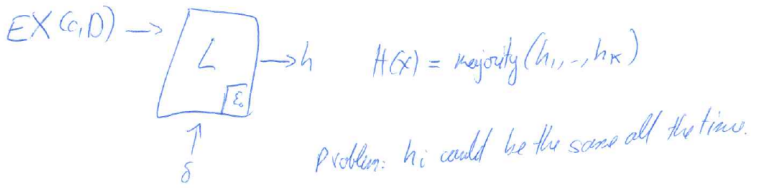


Figure 10: Majority vote.

The problem is that L could be evil and always output the same hypothesis. The key idea is to create filtered distributions to force L to learn something "new".

8.1 "Original" Boosting construction (Schapire)

1. Call weak learning algorithm L on $D_1 = D$ and we get h_1 such that $Err_{D_1}(h_1) \leq \varepsilon_0$
2. We create a new distribution, D_2. To sample from D_2:
   - Flip a fair coin.
   - If heads, sample $x \sim D_1$ until $h_1(x) \neq c(x)$.
   - If tails, sample until $h_1(x) = c(x)$.

   Remark 8.2. Note that if these conditions are not met for a lot of samples, then either h_1 or $\neg h_1$ are already good enough hypothesis. This guarantees that L cannot output h_1 or its negation as hypothesis because they would have error 50/50.

   Algebraically, this corresponds to modifying the weights of the original distribution, see the picture below.

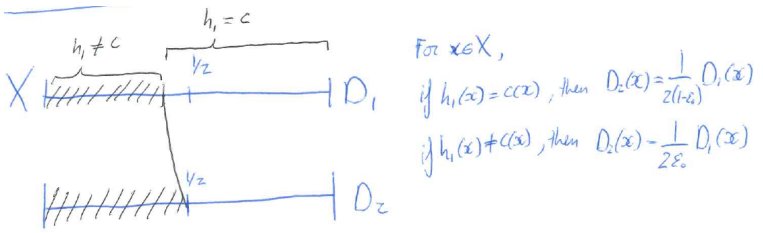


Figure 11: Modify weights of original distribution.

   Then we run L on $EX(c, D_2)$ to get some $h_2 \neq h_1$ such that $Err_{D_2}(h_2) \leq \varepsilon_0$.
3. Define a distribution D_3. To sample from D_3:
   - Draw $x \sim D_1$ until $h_1(x) \neq h_2(x)$. We will quickly get such a an x by construction.
   - Create labeled example $\langle x, c(x)\rangle$.

   Run L on $EX(c, D_3)$ to get h_3 such that $Err_{D_3}(h_3) \leq \varepsilon_0$.
4. Final hypothesis, for any x, $h(x) = \text{majority}\{h_1(x), h_2(x), h_3(x)\}$.

Lemma 8.3. $Err_{D_1}(h) \leq 3\varepsilon_0^2 - 2\varepsilon_0^3$.

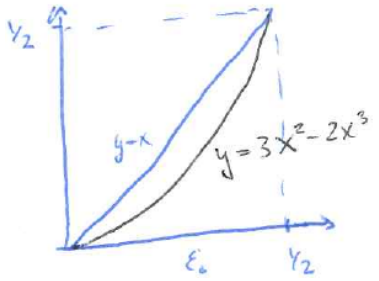


Figure 12: $3x^2 - 2x^3$.

Remark 8.4. Note that it is a convex function below the line $\{y = x\}$. So we gain a little bit by boosting. Then we can iterate to obtain an arbitrary boosting.

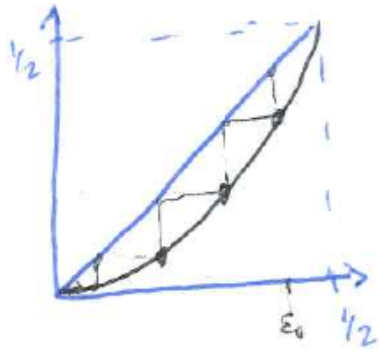


Figure 13: Boosting iterations.

Also note that the number of iterations needed to obtain error $\leq \varepsilon$ is $\sim \log\log\frac{1}{\varepsilon}$.

We are actually changing the hypothesis space in the final algorithm. We output a ternary tree of majority hypotheses

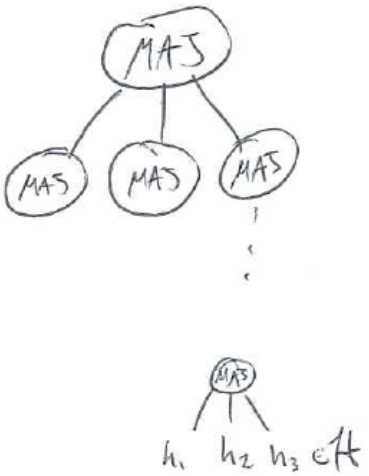


Figure 14: Majority tree.

8.2 Adaboost (Freund/Schapire)

- View input $S = (x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m) \sim EX(c, D)$. Assume that $y_i \in \{-1, 1\}$. We want to use the weak learning algorithm to find a hypothesis that is consistent with S, recall that as we saw before this is enough for PAC learning.
- Start with D_1 the uniform distribution on S. If we can get h such that $Err_{D_1}(h) < \frac{1}{m}$, then h is consistent with S and apply VC-theory.
- Let's look at the "code", recall that the distributions refer to S.

  $D_1 = \text{Unif}(S)$

  for t in range $1, \ldots, T$

  - run weak learning algorithm using D_t to get some $h_t \in \mathcal{H}$
  - choose a weight $\alpha_t >= 0$ for the hypothesis h_t. (The appropriate choice is $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$, see analysis below).
  - define the next distribution D_{t+1}

  $$D_{t+1}(x_i, y_i) = D_t(x_i, y_i) e^{-\alpha_t y_i h_t(x_i)} Z_t, \tag{8.2}$$

  where Z_t is the normalization factor.
  Remark 8.5. Note that $y_i h_t(x_i)$ is 1 if they agree or -1 otherwise. This means that we are increasing the weight on the places where h_t was wrong and reducing the weight on the ones where it was right.

  Final classifier: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Analysis

Remark 8.6 (Notation). Let $\varepsilon_i \coloneqq \mathbb{P}_{i \sim D_t}[h_t(x_i) \neq y_i]$ and $\gamma_t \coloneqq \frac{1}{2} - \varepsilon_t$ "advantage" of h_t over random guessing.

If $y_i \neq H(x_i) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x_i)\right)$, then $y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) \leq 0$, which implies that $e^{-\sum_{t=1}^{T} \alpha_t h_t(x_i)} \geq 1$. So

$$\frac{1}{m} \left|\{i : H(x_i) \neq y_i\}\right| \leq \frac{1}{m} \sum_i e^{-\sum_t \alpha_t h_t(x_i)}. \tag{8.3}$$

Take any <u>fixed</u> i, we can measure how much the distribution changes from t to $t+1$

$$Z_t = \frac{D_t(x_i, y_i) e^{-\alpha_t y_i h_t(x_i))}}{D_{t+1}(x_i, y_i)} \tag{8.4}$$

Now we want take a look at how much the distribution has changed over t

$$\prod_t^T Z_t = \frac{D_1(x_i, y_i)}{D_{T+1}(x_i, y_i)} e^{-y_i \sum_t \alpha_t h_t(x_i)} = e^{-y_i \sum_t \alpha_t h_t(x_i)}, \tag{8.5}$$

since the Z_t's are independent of i so must be the right-hand-side above. We obtain,

$$\prod_t^T Z_t = \frac{m}{m}\prod_t^T Z_t = \frac{1}{m}\sum_i e^{-y_i \sum_t \alpha_t h_t(x_i)}, \tag{8.6}$$

so to estimate the error we only need to analyze the Z_t's and recall that we are still free to choose the weights α_t's

$$Z_t = \sum_i D_t(x_i, y_i) e^{-\alpha_t y_i h_t(x_i)}, \tag{8.7}$$

therefore, we can choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ to obtain

$$Z_t = (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t} = \cdots \leq 2\sqrt{\varepsilon_t(1-\varepsilon_t)}. \tag{8.8}$$

Thus, we can bound the training error

$$\text{training error} \leq \prod_t 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = \prod_t 2\sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t}. \tag{8.9}$$

We have arrived at the following result.

Theorem 8.7. *Our training error on the original S is bounded above by $e^{-2\sum_t \gamma_t}$.*

Remark 8.8 (Out of sample generalization). In particular, if we know that $\gamma_t \geq \gamma$, we have that a simpler bound for the training error: $\frac{1}{m}\left|\{i : H(x_i) \neq y_i\}\right|$. Then if we choose the number of rounds of Adaboosting, $T \geq \frac{1}{2\gamma^2}\ln(m)$, we can guarantee that H is consistent with the set S. Further, if $VCD(\mathcal{H}) = d$, then the VCD of $H's$ class is $\leq dT$ (linearity result for VCD-dimension, check the textbook). This means that if we choose T so that $T \geq \frac{1}{2\gamma^2}\ln(dT)$, then by VCD-theory we have good generalizations out of the sample S, i.e. with respect to the true distribution.

Remark 8.9. Adaboost provides a sort of converse to the following result: "hypothesis compression" $\implies$ "learning". If we have an algorithm that has a bad relation with the ε of the PAC-learning model, i.e. it outputs big hypothesis with respect to the error, then we can choose a bigger ε and feed this "bad" algorithm to obtain a much better hypothesis (In fact, sub-linear in ε).

9 Lecture 9: 2017.03.27

Today

- Learning with queries
- Algo for DFA
- PAC "solar system"

9.1 Learning with queries

PAC + Membership Queries

In essence, it is PAC-learning with a membership query oracle. This gives the algorithm to create "synthetic" examples and ask for the true labels. This assumes that there is an underlying ground truth, which might not be true in practice.

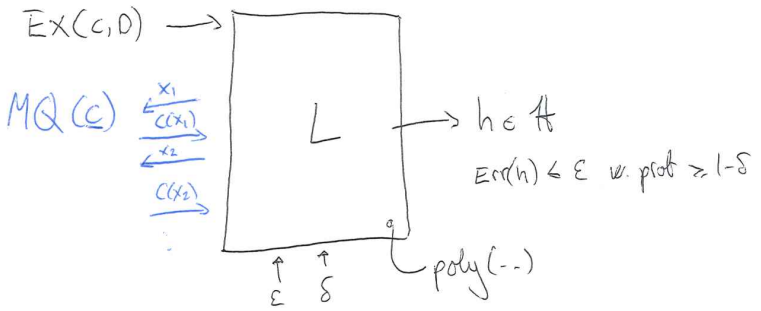


Figure 15: PAC with membership queries.

Remark 9.1. On the surface, adding these membership queries are adding some power to the algorithm that cannot be simulated with $EX(c, D)$ or statistical queries.

Exact Learning with Membership & Equivalence Queries

Now consider the following learning model where we have membership queries of individual inputs and an equivalence query oracle that answers whether an hypothesis is the same as the target class (as functions), c, and gives *some* counterexample if appropriate i.e. some input where they disagree.

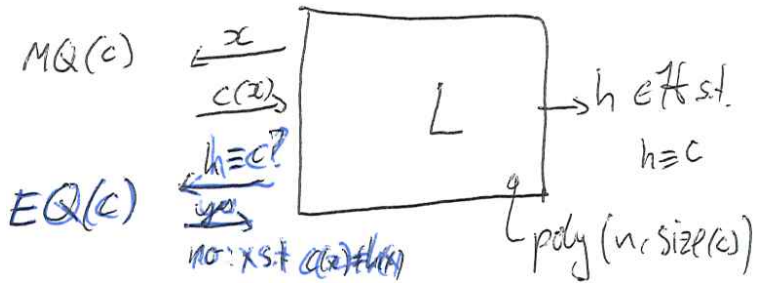


Figure 16: Exact learning with membership and equivalence queries.

Theorem 9.2. *If $\mathcal{C}$ is exactly learnable from MQ & EQ, then $\mathcal{C}$ is PAC+MQ-learnable.*

Sketch of proof. We can approximate EQ with enough samples from EX. In other words, we sample looking for a counterexample. If we find it great, we can return it. If we don't find it in a reasonable time, then also great because we already have a reasonably good hypothesis. □

9.1.1 Exact Learning of DFAs in MQ & EQ (Angluin)

First of all, recall that PAC-learning DFAs[3] is provably intractable assuming that factoring is hard.

Example 9.3 (DFA)**.** Consider $X = \{0,1\}^*$ with labels $\{+,-\}$. i.e. arbitrary length strings of 1s and 0s. The goal is to learn in poly(size(c), longest counterexample).

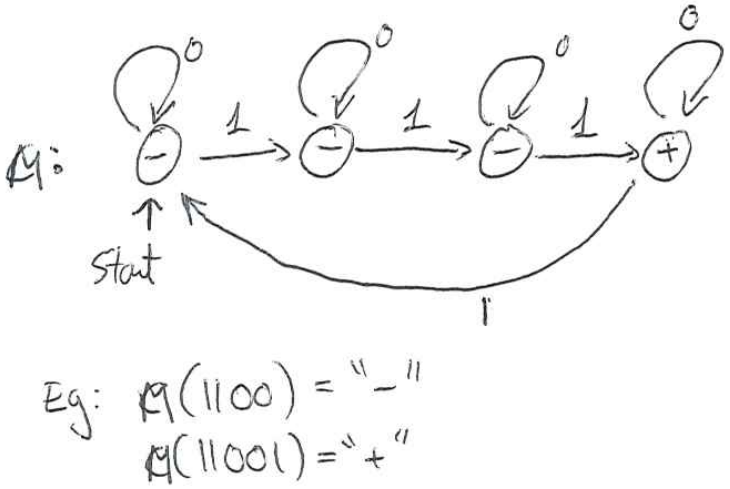


Figure 17: Example of DFA.

This function is actually $M(x) = +$ if and only if the #1s in x is $\equiv 3 \mod 4$.

Data structure: classification tree T

To be able to learn DFAs we use a binary tree satisfying the following properties.

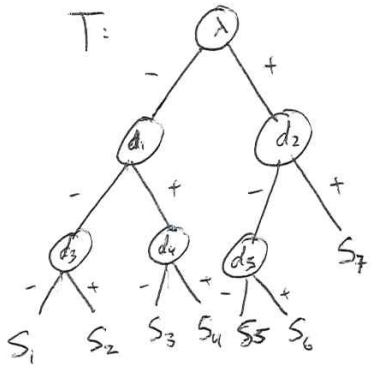


Figure 18: Classification tree.

- All the s and d are strings, λ is the empty string.

[3]Deterministic Finite Automata.

- Leaves s: state access strings.
- Internal nodes d: distinguishing strings.
- Invariants:
  - i) Each s_i reaches a different state of M.
  - ii) For all s_i and d_j, s_i is in left/right subtree of $d_j \iff s_i d_j$ reaches a $-/+$ state of M.

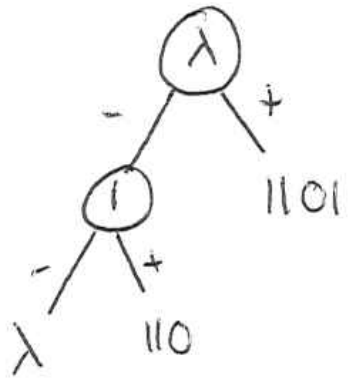


Figure 19: Classification tree for the example above.

Learning Model

1. MQ on λ to get the start state of λ i.e. what is the label of the start state, say $+$.
2. EQ on the trivial machine:

Figure 20: Trivial DFA.

If the machine is not trivial, we obtain a counterexample string, $\langle s, - \rangle$, and we can initialize the tree.

Figure 21: Initial classification tree.

3. Now we can iterate this process using equivalence queries and enlarging the tree. But before we can do that we need how to transform a classification tree into a hypothesis DFA and how to make progress on the tree given a new counterexample.

   Definition 9.4 (Sift operation)**.** Given a classification tree, T, and a string of 1s and 0s, s, we define the sift of s by T, $\text{sift}(T, s)$, as the leaf that is reached by the string s in the following way. At each node, we follow the string s and then the string of the node d_i in the true DFA machine and depending on the label of the state we reach, we go left/right down the tree. In the learning model we can actually do this using membership queries. For example:

   - At the root, MQ the string $s\lambda$. Say that the query returns $+$, that means we go down the right node.
   - MQ the string sd_2 and suppose the query returns $-$, we go down the left node.
   - Repeat until we reach a leaf of the tree.

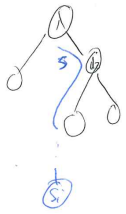


Figure 22: Sift operation.

Remark 9.5.

- A tree T produces a partition of the space of all strings by labeling each string with $\text{sift}(T, s)$.
- Strings that reach the same state o the true machine must sift to the same leaf. This is because of the properties of the tree mentioned above.

From a tree to a hypothesis machine $\widehat{M}$.

- For each leaf of the tree we have a state of the machine.
- To obtain the transitions between states we use the sift operation. For example, the state corresponding to the leaf s_i is connected to $\text{sift}(T, s_i 0)$ and $\text{sift}(T, s_i 1)$.

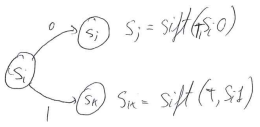


Figure 23: Transitions between states of $\widehat{M}$.

Making progress on the classification tree.
Once we have $\widehat{M}$ we can EQ and either be done or produce a counterexample. Suppose that the equivalence query on $\widehat{M}$ produces the counterexample $\gamma = \gamma_1\gamma_2\ldots\gamma_m$ where $\gamma_l \in \{0,1\}$. For simplicity assume that γ is $+$ in the true machine M and $-$ in $\widehat{M}$. We use γ to improve the classification tree.

- As observed above, our current tree induces a partition of the space of strings, $\{0,1\}^*$, and, equivalently, on the states of the machine M.
- Each of the partition pieces corresponds to one of the current leaves.
- We can assume that $S_1 = \lambda$, i.e. that the first leaf tracks the start state of the machine.
- Follow γ under $\widehat{M}$ and M and see where they diverge.

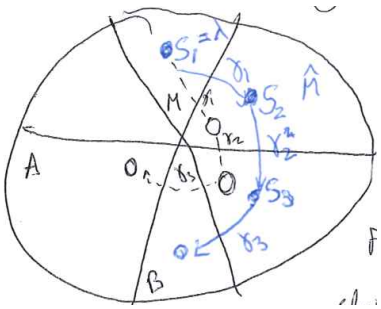


Figure 24: Split criterion for leaves.

Since they diverge, at some point we must end up in different equivalence classes which implies that on the step immediately before that we have two different states on the same partition piece. In the figure above, this happens after following $\gamma_1\gamma_2$. Thus, the string $\gamma_1\gamma_2$ reaches a *new state* that used to be in the same equivalence class as S_3. We can use this to split the leaf S_3.

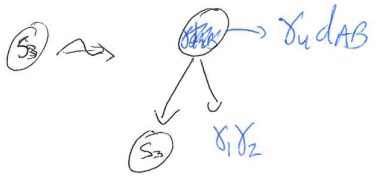


Figure 25: Splitting a leaf.

where d_{AB} is the distinguishing string of the least common ancestor of the leafs A and B, which are also leaves of our tree.

4. The stopping condition is when we have as many leaves as the true machine has states.

9.2 PAC-learning "Solar System"