1 Lecture 2: 2017.01.30

Remark 1.1 (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 1.2. *The class of rectangles over $\mathbb{R}^2$ from example above is PAC learnable (with sample size $m \sim \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)$).*

1.1 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 $c = x_1 \wedge \neg x_3 \wedge \neg x_{11} \wedge x_{26} \ldots$,

$$c(\vec{x}) = 1 \iff x_1 = 1, x_3 = 0, x_{11} = 0, x_{26} = 1, \ldots \tag{1.1}$$

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

1.1.1 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,

$$p_i = \mathbb{P}_{\bar{X} \sim D}[c(\vec{x}) = 1, x_i = 0]. \tag{1.2}$$

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

$$Err(h) \leq \sum_{x_i \in h} p_i \leq n \left(\frac{\varepsilon}{n}\right). \tag{1.3}$$

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

$$\mathbb{P}[x_i \text{ "survives" m random samples}] = (1-p_i)^m \quad (iid) \tag{1.4}$$

$$\leq \left(1 - \frac{\varepsilon}{n}\right)^m \tag{1.5}$$

$$\mathbb{P}[\textit{any} \text{ bad i "survives" m random samples}] \leq n\left(1 - \frac{\varepsilon}{n}\right)^m. \tag{1.6}$$

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

Remark 1.4. *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 1.5. *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.*

1.2 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.

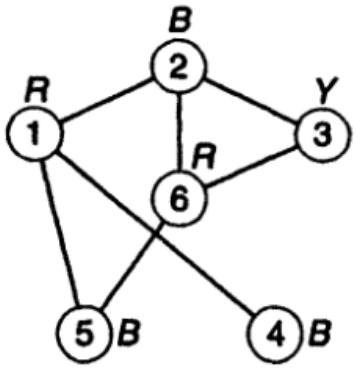

Figure 1: A graph with a 3-coloring.

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

$$T_R = x_2 \wedge x_3 \wedge x_4 \wedge x_5, \tag{1.7}$$

$$T_B = x_1 \wedge x_3 \wedge x_6, \tag{1.8}$$

$$T_Y = x_1 \wedge x_2 \wedge x_4 \wedge x_5 \wedge x_6. \tag{1.9}$$

Claim 1.6. *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 $\frac{1}{\#examples}$[1]. 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 1.7. *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 1.8. *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 $NP = RP$)*

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