CSE 513T Theory of Artificial Intelligence & Machine Learning Feb 1, 2018

Lecture 6

Lecturer: Brendan Juba *Scribe: Jiarui XU*

Contents

1 Warm-up Problem **1**

2 PAC-learning with Occam's Razor **1**

3 Decision List **2**

1 Warm-up Problem

Problem

How many bits does it take to specify functions represented by arbitrary conjuctions on n attributes?

Solution

$\#\{\text{b-bit strings}\} = 2^b + 2^{b-1} + \cdots + 2^0 = 2^{b+1} - 1$. Excluding contradictions, 3 choices per i^{th} attribute, x_i, $\neg x_i$, or missing. So that total number of conjuctions is 3^n + all-false "contradiction". So we would like to find the minimum b s.t.

$$\begin{gathered} 2^{b+1} - 1 \geq 3^n + 1 \\ 2^{b+1} \geq 3^n + 2 \\ b + 1 \geq \log_2 3^n + 2 \\ b + 1 \geq \lceil \log_2 3^n + 2 \rceil \\ \text{asymptotics } b \sim n \cdot \log_2 3 \end{gathered}$$

2 PAC-learning with Occam's Razor

Occam: "Entities must not be multiplied without necessity" or prefer the simpler theory seems vague add ill-defined... We'll see nevertheless in the "PAC" model of inductive learning Occam's Razor is a useful theorem.

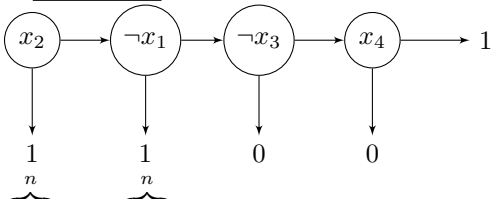
Recall $Consis_{\mathcal{H}}^{B(n,m)}$ given a list of labeled examples $\begin{bmatrix} \bar{x}^{(1)}, b^{(1)} \\ \bar{x}^{(2)}, b^{(2)} \\ \vdots \\ \bar{x}^{(m)}, b^{(m)} \end{bmatrix}$ on n attributes, find a representation $c \in \mathcal{H}$ consistent with the example and with $|c| \leq B(n, m)$. An "Occam Algorithm" for B(n, m) is any algorithm solving this search problem.

Theorem (Occam's Razor)**.** *There is a constant $b > 0$ such that for any class of representations $\mathcal{C}_n$ on n attributes and any distribution on n-attribute examples if the number of examples m is such that $B(n, m) \leq b\epsilon m - \log_x \frac{1}{\delta} - 1 \Leftrightarrow m \geq \frac{1}{b\epsilon}(B(n.m) + \log_2 \frac{1}{\delta} + 1)$ Then with probability $1 - \delta$, any solution h to $Consis_{\mathcal{H}}^{B(n,m)}$ on the data set satisfies $Pr_{\bar{x} \in \mathcal{D}}[h(\bar{x}) = c(x)] \geq 1 - \epsilon$. Thus, any "Occam Algorithm" that runs in time polynomial in n, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, size(c) is a PAC-Learning algorithm for $\mathcal{C}$, as long as the bound on $B(n, m)$ is satisfied.*

Proof. We say that $h \in \mathcal{H}$ is "bad" if $Pr_{\bar{x}\in\mathcal{D}}[h(\bar{x}) \neq c(x)] > \epsilon$. Then the probability that a bad $h \in \mathcal{H}$ is consitent with m examples is $(1-\epsilon)^m$. For m satisfying the given condition, the probability is at most $(1-\epsilon)^{\frac{1}{b}\cdot\frac{1}{\epsilon}\cdot(B(n,m)+\log_2\frac{1}{\delta}+1)}$ Since $(1-\epsilon)^{\frac{1}{\epsilon}} \leq e^{-1}$, this is at most $e^{-\frac{1}{b}(B(n,m)+\log_2\frac{1}{\delta}+1)}$. For $b = \frac{1}{ln2}$, this becomes $2^{-(B(n,m)+\log_2\frac{1}{\delta}+1)} = \frac{\delta}{2^{B(n,m)+1}}$. Since the solution to $Consis_{\mathcal{H}}^{B(n,m)}$ can be represented by $\leq B(n,m)$ bits, there are fewer than $2^{B(n,m)+1}$ possible h for the algorithm to output. So the probability that any solution to $Consis_{\mathcal{H}}^{B(n,m)}$ is "bad" is at most $2^{B(n,m)+1} \cdot \frac{\delta}{2^{B(n,m)+1}} = \delta$ by a union bound. $\Rightarrow$ with probability $1-\delta$ any solution to $Consis_{\mathcal{H}}^{B(n,m)}$ is not bad. □

3 Decision List

A decsion list is a decision tree in which one branch always leads to a leaf.

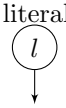


Theorem. *There is a (polyomial time) proper PAC-learning algorithm for decision list.*
Note: since (observation), the list need never have length $> 2n$, and we can write a node of the list using $\approx \log_2 n + 2$ bits. So we can use $B(n,m) = O(nlogn)$ and simply find a consistent decision list.
For a literal l and a set of examples S, let $S_l = \{(\bar{x}, b) \in S, l(\bar{x}) = 1\}$. We will say that l is "useful for" S if for every $(\bar{x}, b) \in S_l$, b is the same, and S_l is not $\varnothing$.

Proof. For the analysis, it suffices to show on each iteration, the algorithm finds a useful literal (since the size of S decreases on each iteration, the loop ends when S has ≤ 1 example in it).

Algorithm 1 FindList(S)

procedure FINDLIST
Let h be the empty list
while n **do**ot every example in S has the same label
Iterate over the literals until a useful literal l is found.



For the label b of examples in S_l, add to end of h.
$S \leftarrow S - S_l$.
For the common label b of every remaining example in S, add the terminal leaf node $\rightarrow b$ to h.
Return h

We note that when we remove an example from S, it will be labeled correctly by h at the node we added at the iteration where we remove the example.

Remark Why does a useful literal l always exist?
Note that we assume that the examples are actually labeled by some decision list c. In general, if we consider the first literal of the unknown c not included in h, all of the remaining examples must be consistent with that literal. Either some examples are included that satisfy the literal l, and thus l is useful, or the remaining examples must be consitently labeled by next literal, etc. By induction, the first literal of c that has examples remaining in S and is not included in h is always useful. If no such

literal exists, it is because none of the literals of c pick up any examples in S, and then they all reach the tail of c, and thus all have the same label. $\square$

Much as how we extened the Elimination algorithm to learn k-CNFs and k-DNFs, we can find Decision lists over attributes given by conjunctions of k attributes, call these "k-decision lists". Our algorithm takes $O(\frac{1}{\epsilon}(kn^k log n + log\frac{1}{\delta}))$ examples, still runs in time $O(m \cdot n^k)$ still polyomial.