TTIC 31250 : Introduction to Theory of Machine Learning

Spring 2020

•

Homework 4

•

Solutions by

Jinhong Du

dujinhong@uchicago.edu

12243476

Problems

1. **PAC-learning of small OR-functions.** Suppose the target function is a disjunction (OR-function) of r out of n boolean variables, for r is much less than n. For example, perhaps only $\sqrt{n}$ of the n variables are used in the target ($r = \sqrt{n}$). The list-and-cross-off algorithm for learning an OR function might produce a hypothesis of size $O(n)$, so its sample size for PAC-learning would be $\tilde{O}\left(\frac{n}{\epsilon}\right)$. Alternatively, we could try all $O(n^r)$ possible OR-functions of size at most r and pick one consistent with our training sample. Since $\log|\mathcal{H}| = r\log n$, this would require only $\tilde{O}\left(\frac{r\log n}{\epsilon}\right)$ samples but takes time exponential in r.

   Your job: Give a polynomial-time algorithm that guarantees to find an OR of at most $O(r\log m)$ variables that is consistent with the training data, where m is the number of training examples. So this is not quite finding the *smallest* OR-function, but its close. Describe a variant of your algorithm that runs in polynomial time and finds an OR of only $O\left(r\log\left(\frac{1}{\epsilon}\right)\right)$ variables, with error at most $\frac{\epsilon}{2}$ on the training data. Note that by Occam + Chernoff bounds, the latter algorithm requires sample size at most $O\left(\frac{1}{\epsilon}\left(r\log\frac{1}{\epsilon}\log n + \log\frac{1}{\delta}\right)\right)$ to learn in the PAC model, so is just very slightly worse than the exponential-time algorithm that tries all $O(n^r)$ OR-functions of size at most r.

   Hint: think about the greedy set-cover algorithm (look it up if you haven't seen it).

   Proof. We denote the n variables by $x_1,\ldots,x_n$, the m samples by $\mathcal{S} = \{(X_i, Y_i) : X_i \in \{0,1\}^n,\ Y_i \in \{0,1\},\ i = 1,\ldots,m\}$ and $X_{i,j}$.
   The greedy algorithm is given by

   (1) Initialize the set of all single terms $\mathcal{T}_0 = \{x_1,\ldots,x_n,\overline{x}_1,\ldots,\overline{x}_n\}$, and the hypothesis $h_0 = -$.

   (2) For $t = 1, 2, \ldots$ do

   i. If $err_{\mathcal{S}}(h_{t-1}) = 0$, stop loop and return h_{t-1}.

   ii. Pick $x \in \mathcal{T}_{t-1}$ such that $\sum_{\{i:Y_i=0\}} err_i(x) = 0$ and

   $$x = \arg\min_{x\in\mathcal{T}_{t-1}} err_{\mathcal{S}}(h_{t-1} \vee x).$$

   iii. Let $h_t = h_{t-1} \vee x$ and $\mathcal{T}_t = \mathcal{T}_{t-1} \setminus \{x\}$.

   Note that h_t produced by the above algorithm will never make mistakes on negative sample.
   If $Y_1 = \cdots = Y_m = 0$, we just return the default hypothesis $h = -$. Otherwise, we will have p $(1 < p < m)$ positive samples in $\mathcal{S}$. Since the target OR function only contains r terms, a positive sample will be correctly classified if and only if at least one of these terms predict it correctly. So the first term picked by the greedy algorithm correctly classifies at least $(m-p) + \frac{p}{r}$ samples. So the set of misclassified samples $\mathcal{S}_1$ after step 1 will satisfy

   $$|\mathcal{S}_1| \le p\left(1 - \frac{1}{r}\right).$$

   Then at least one of $\mathcal{T}_1$ must correctly classifies $\frac{|\mathcal{S}_1|}{r-1}$ samples in $\mathcal{S}_1$, because otherwise the optimum solution would have to contain more than r terms.
   Suppose that after $(t-1)$th step, the set of remaining misclassified samples is $\mathcal{S}_{t-1}$. Then at the tth step,

   (a) If we have already included r terms of the target function to h_{t-1}, the algorithm will halt.

Solution (cont.)

(b) Otherwise, there are still some terms of the target function in $\mathcal{T}_{t-1}$. As we know that there is a cover of size r for these examples (namely, the target OR function), and so by the pigeonhole principal there exists some terms in $\mathcal{T}_{t-1}$ that predict correctly at least $\frac{|\mathcal{S}_{t-1}|}{r}$ samples in $\mathcal{S}_{t-1}$, which yields

$$|\mathcal{S}_t| \leq |\mathcal{S}_{t-1}| \left(1 - \frac{1}{r}\right).$$

Therefore, in general, if the algorithm continues at step t, we will have

$$|\mathcal{S}_t| \leq |\mathcal{S}_{t-1}| \left(1 - \frac{1}{r}\right) \leq m \left(1 - \frac{1}{r}\right)^t.$$

Let $m\left(1 - \frac{1}{r}\right)^t < 1$, we have

$$\begin{aligned} e^{-\frac{t}{r}} \approx \left(1 - \frac{1}{r}\right)^{r\frac{t}{r}} &< \frac{1}{m}, \\ \frac{r}{t} &> \log m, \\ t &< r \log m, \end{aligned}$$

which ensures that the algorithm will choose at most $O(r \log m)$ terms. Furthermore, as in each step we only need to evaulate at most m points, this algorithm only takes time in polynomial.
To extend the above algorithm, we stop it if $err_{\mathcal{S}}(h_{t-1}) < \frac{\epsilon}{2}$ rather that $err_{\mathcal{S}}(h_{t-1}) = 0$. Analogously, let $m\left(1 - \frac{1}{r}\right)^t < \frac{\epsilon}{2}m$, we have

$$\begin{aligned} e^{-\frac{t}{r}} \approx \left(1 - \frac{1}{r}\right)^{r\frac{t}{r}} &< \frac{\epsilon}{2} \\ \frac{r}{t} &> \log \frac{2}{\epsilon}, \\ t &< r \log \frac{2}{\epsilon}, \end{aligned}$$

i.e., we only need $O\left(r \log\left(\frac{1}{\epsilon}\right)\right)$ variables. □

2. **Uniform distribution learning of DNF Formulas.** Give an algorithm to learn the class of *DNF formulas* having at most s terms over the *uniform distribution* on $\{0,1\}^n$, which has sample size polynomial in n and s, and running time $n^{O(\log \frac{s}{\epsilon})}$. So, your algorithm matches the SQ-dimension lower bounds.

   Hint: Think of your algorithm for Problem 1.

   Note: your solution requires that data come from the uniform distribution. The best algorithm known for learning polynomial-size DNF formulas over general distributions has running time roughly $2^{O(n^{\frac{1}{3}})}$ [Klivans-Servedio].

Proof. The probability of a random positive sample $(X,1)$ from $\mathcal{D}$ the uniform distribution over $\{0,1\}^n$ satisfies a conjuction $x_{i_1} \wedge \cdots \wedge x_{i_L}$ with size L is

$$\mathbb{P}_{X\sim\mathcal{D}}(X_{i_1} = \cdots = X_{i_L} = 1) = \frac{1}{2^L}.$$

And the case when the conjuction contains negation of some variables is analogous. So the probability of such a random positive sample satisfies a conjuction with size less than or equal to L is

$$\sum_{l=1}^{L} \frac{1}{2^l} = 1 - \frac{1}{2^L}.$$

Let $1 - \frac{1}{2^L} \geq 1 - \frac{\epsilon}{s}$, we have

$$L \geq \log_2 \left(\frac{s}{\epsilon}\right).$$

Therefore, we just need to choose conjuctions with size at most $L = \lceil \log_2 \left(\frac{s}{\epsilon}\right) \rceil$. Then with probability at least $1-\epsilon$, each term/conjuction of the target DNF formual would have at most L variables. At the algorithm in Problem 1, the set of covers are the set of all terms, while in this case, the set of covers $\mathcal{T}_0$ will be the set of totally $\sum_{l=1}^{L} \binom{2n}{l} = O(n^L)$ conjuctions with size less than or equal to L. So the algorithm will be given by

(1) Initialize $\mathcal{T}_0$ the set of all conjuctions with size less than $\lceil \log_2 \left(\frac{s}{\epsilon}\right) \rceil$, and the hypothesis $h_0 = -$.

(2) For $t = 1, 2, \ldots$ do

   i. If $err_{\mathcal{S}}(h_{t-1}) \leq \epsilon$, stop loop and return h_{t-1}.

   ii. Pick $x \in \mathcal{T}_{t-1}$ such that $\sum_{\{i:Y_i=0\}} err_i(x) = 0$ and

$$x = \arg\min_{x\in\mathcal{T}_{t-1}} err_{\mathcal{S}}(h_{t-1} \vee x).$$

   iii. Let $h_t = h_{t-1} \vee x$ and $\mathcal{T}_t = \mathcal{T}_{t-1} \setminus \{x\}$.

Note that we requires that data come from the uniform distribution, so that the stop criteria $err_{\mathcal{S}}(h_{t-1}) \leq \epsilon$ can be obtained within $O(L \log \left(\frac{1}{\epsilon}\right))$ steps.
As we need to evaulate $|\mathcal{T}_t|$ conjuctions on $|\mathcal{S}| = poly(n,s)$ samples and this will stop after $O(L \log \left(\frac{1}{\epsilon}\right))$ steps, this algorithm will take time at most

$$O\left(n^L |\mathcal{S}| \cdot L \log\left(\frac{1}{\epsilon}\right)\right) = n^{O(\log(\frac{s}{\epsilon}))}.$$

□

3. **SQ learning Decision Lists and Trees.** Give an algorithm to learn the class of *decision lists* in the SQ model (and argue correctness for your algorithm). Your algorithm should work for any distribution $\mathcal{D}$ (not just the uniform distribution). Be clear about what specifically the queries χ are and the tolerances τ. Remember, you are not allowed to ask for conditional probabilities like $\Pr[A|B]$ but you can ask for $\Pr[A \wedge B]$.

   So, combined with your results from Homework 1, this gives an algorithm for learning decision trees of size s in the SQ model with $n^{O(\log s)}$ queries of tolerance $\frac{1}{n^{O(\log s)}}$, matching our SQ-dimension lower bounds.

   Note: this problem can be a bit tricky. In particular, it is possible to create a distribution $\mathcal{D}$ over $\{0,1\}^n$ and a target decision list c with the following properties:

   (a) $\Pr_{\mathcal{D}}[c(x) = 1] = \frac{1}{2}$.

   (b) For all $1 \leq i \leq n$, either $\Pr_{\mathcal{D}}[x_i = 1] \leq 2^{-\frac{n}{2}}$ or else $\Pr_{\mathcal{D}}[c(x) = 1 | x_i = 1] = \frac{1}{2}$.

   In particular, no variable is noticeably correlated with the target! (Any variable either is almost never 1 or else is completely uncorrelated with the target). So, an algorithm based on trying to find x_i, y, b such that (a) $\Pr[c(x) = y | x_i = b]$ is large and (b) "$x_i = b$" happens with noticeable probability is going to have trouble. In fact, this difficulty is one reason that there is no known analog of Problem 1 for decision lists. Instead, think about building on the algorithm from the very first lecture.

Proof. Let $c = \langle (x_{t_1^*} = v_1^*, l_1^*), \ldots, (x_{t_r^*} = v_r^*, l_1^*), (true, l_{r+1}^*) \rangle$ be the target decision list. We use $\overline{v}$ to denote the negation of v.

(1) Initialize the indexes of all variables

$$\mathcal{T}_0 = \{1, \ldots, n\},$$

h_0 as a empty decision list, and step $i = 1$.

(2) Repeat

(i) For $l \in \{0,1\}$, make queries about

$$f_i(l) = \mathbb{P}_{x \in \mathcal{D}}(x_{t_1} = v_1 \wedge \cdots \wedge x_{t_{i-1}} = v_{i-1} \wedge c(x) = l),$$

and recieve $\hat{f}_i(l)$ with tolerance $\tau_f = \frac{\epsilon}{2(n+1)}$. If there exists a $l \in \{0,1\}$ such that $\hat{f}_i(l) \leq \frac{\epsilon}{2(n+1)}$, then append $(true, \bar{l})$ to h_{i-1}, return it and halt.

(ii) For $t \in \mathcal{T}_{i-1}$, make quires about

$$p_i(t, v, l) = \mathbb{P}_{x \in \mathcal{D}}(x_{t_1} = v_1 \wedge \cdots \wedge x_{t_{i-1}} = v_{i-1} \wedge x_t = v \wedge c(x) = l).$$

and recieve answer $\hat{p}_i(t, v, l)$ with tolerance $\tau_p = \frac{\epsilon}{2(n+1)}$. Find a pair $(t_i, v_i, l_i) \in \mathcal{T}_{i-1} \times \{0,1\} \times \{0,1\}$ such that

$$(t_i, v_i, l_i) = \underset{(t,v,l) \in \mathcal{T}_{i-1} \times \{0,1\} \times \{0,1\}}{\arg\min} \hat{p}_i(t, v, \bar{l}),$$

and

$$\hat{p}_i(t_i, v_i, \bar{l}_i) \leq \frac{\epsilon}{2(n+1)}.$$

Append $(x_{t_i} = v_i, l_i)$ to h_{i-1} as the ith item to get a new decision list h_i.

(iii) Set $\mathcal{T}_i \leftarrow \mathcal{T}_{i-1} \setminus \{t_i\}$ and $i \leftarrow i + 1$.

Let r_h be the length of h, which is produced by the algorithm. For the above algorithm, we have the following results:

Solution (cont.)

1. The queries we make in the algorithm are

$$\chi_{f_i(l)} = (x_{t_1} = v_1 \wedge \cdots \wedge x_{t_{i-1}} = v_{i-1} \wedge c(x) = l), \qquad i = 1, \ldots, r_h + 1$$
$$\chi_{p_i(t,v,l)} = (x_{t_1} = v_1 \wedge \cdots \wedge x_{t_{i-1}} = v_{i-1} \wedge x_t = v \wedge c(x) = l), \qquad i = 1, \ldots, r_h$$

   for $v, l \in \{0, 1\}$. The tolerances are $\tau_f = \tau_p = \frac{\epsilon}{2(n+1)}$.

2. We are guaranteed that

$$p_i(t_i, v_i, \bar{l}_i) \leq \hat{p}_i(t_i, v_i, \bar{l}_i) + \tau_p = \frac{\epsilon}{n+1},$$

   for $i = 1, \ldots, r_h$, and

$$f_i(\bar{l}_{r_h+1}) \leq \hat{f}_{r_h+1}(\bar{l}_{r_h+1}) + \tau_f = \frac{\epsilon}{n+1}.$$

3. **All rules of c occur in h, or otherwise the default rule of h always fires correctly.** First, since

$$\hat{p}_1(t_1^*, v_1^*, \bar{l}_1^*) \leq p_1(t_1^*, v_1^*, \bar{l}_1^*) + \tau_p = \frac{\epsilon}{2(n+1)},$$

   the term $x_{t_1^*} = v_1^*$ or $x_{t_1^*} = \overline{v}_1^*$ must exist in h (it can correctly fire for all examples and any subset of all examples). If the latter happens, then the step (i) in the algorithm will defaultly output l_1^*, which is trivial. So we consider the case when $x_{t_1^*} = v_1^*$ exists in h. So the samples after this rule will fire $x_{t_2^*} = v_2^*$ correctly, which means that $x_{t_2^*} = v_2^*$ or $x_{t_2^*} = \overline{v}_2^*$ must also exist in h (maybe before of after the rule containing $x_{t_1^*}$). Now suppose that for $i \leq j - 1$, $x_{t_i^*} = v_i^*$ or $x_{t_i^*} = \overline{v}_i^*$ is in h, then analogously either h correctly classifies samples after the rule $x_{t_i^*} = \overline{v}_i^*$ for some i, or $x_{t_j^*} = v_j^*$ or $x_{t_j^*} = \overline{v}_j^*$ is in h. By induction, the result follows.

4. If all rules $x_{t_i^*} = v_i^*$ of c are in h, then the set of samples on which the default rule of h fires is a subset of the set of samples on which the default rule of c fire. This is because that the rules $x_{t_i^*} = v_i^*$ can not fire on these samples. So **the default rule of h always fires correctly**.

5. Combine 3 and 4, we know that the algorithm will always stop at step (i).

6. Since the default rule of h always fires correctly, the misclassification rate of h is

$$err_{\mathcal{D}}(h) = \sum_{i=1}^{r_h} p_i(t_i, v_i, \bar{l}_i) + f_i(\bar{l}_{r_h+1}) \leq (n+1) \cdot \frac{\epsilon}{n+1} = \epsilon.$$

□