CS 4540/CS 8803: Algorithmic Theory of Intelligence Fall 2023

Lecture 3: PAC Learning

Instructor: Santosh Vempala *Lecture date: 9/06*

1 Learning the Class of Conjunctions

Consider data of the form $\{x_1, \ldots, x_n\} \in \{0,1\}^n$. A *conjunction* is a function on some subset of the variables $\{x_1, x_2 \ldots, x_n, \bar{x}_1, \bar{x}_2 \ldots, \bar{x}_n\}$, where $\bar{x}_i = 1 - x_i$. It maps x to 1 if every variable in the subset is 1, and 0 otherwise. For example:

$$\begin{aligned} h(x) &= x_1 \wedge x_2 \wedge \bar{x}_5 \\ h(x) &= \bar{x}_3 \wedge \bar{x}_4 \\ h(x) &= x_1 \wedge x_2 \wedge \cdots \wedge x_r \end{aligned}$$

The size of the hypothesis class is 3^n; for each i, either x_i is in the conjunction, $\bar{x}_i$ is in the conjunction, or both are absent.

Any such conjunction can be represented as a linear halfspace, so the tools for learning halfspaces seen in previous lectures apply without modification. Simply sum the variables in the conjunction, and set the threshold to be the number of variables. Using the examples above:

$$\begin{aligned} h(x) &= (x_1 + x_2 + 1 - x_5 \geq 3) \\ h(x) &= (1 - x_3 + 1 - x_4 \geq 2) \\ h(x) &= (x_1 + x_2 + \cdots + x_r \geq r) \end{aligned}$$

If $x_i \in \{0,1\}$, these two ways of writing the function are equivalent.

1.1 Learn-Conj

Let $\mathcal{D}$ be a distribution over $\{0,1\}^n$. Assume that each example is chosen independently from this distribution. Consider the following algorithm for learning an unknown conjunction.

Algorithm 1 LearnConj

Maintain two sets of indices $S = \{x_1, x_2 \ldots, x_n\}$,$\bar{S} = \{\bar{x}_1, \bar{x}_2, \bar{x}_n\}$
for each input $x^{(i)}$ **do**
  if $l(x^{(i)}) = 1$ **then**
    For each $j \in [n]$, remove $\bar{x}_j$ from $\bar{S}$ if $x_j^{(i)} = 1$, and remove x_j from S if $x^{(i)} = 0$
  else
    Do nothing
  end if
end for
Output the conjunction of the remaining literals in S and $\bar{S}$.

At any point, the hypothesis $h(x) = \left(\bigwedge_{i \in S} x_j\right) \wedge \left(\bigwedge_{i \in \bar{S}} \bar{x}_j\right)$ is consistent with all examples seen so far.

Theorem 1. *With $m \geq \frac{1}{\epsilon}(n \log 3 + \log \frac{1}{\delta})$ examples, with probability $1 - \delta$,* LearnConj *will be correct on a new example from $\mathcal{D}$ with probability $1 - \epsilon$.*

Proof. Let $h \in H$ be a hypothesis with $Pr_{x \sim \mathcal{D}}(h(x) \neq l(x)) > \epsilon$. The probability that h is not eliminated after m examples is at most $(1 - \epsilon)^m$.

The probability that any such 'bad h' survives after m examples is at most $3^n(1 - \epsilon)^m$.

Setting $3^n(1 - \epsilon)^m < \delta$ and taking the log of both sides, we see that $m \geq \frac{1}{\epsilon}(n \log 3 + \log \frac{1}{\delta})$ suffices.

□

1.2 Learning Decision Lists

A *decision list* is a generalization of the set of conjunctions. It consists of a series of if-else statements, returning true/false at each step. E.g.,

If $x_1 = 1$, return False; else if $x_4 = 0$, return True; else, return False.

Any conjunction can be written as a decision list. E.g., $x_1 \wedge x_2 \wedge \cdots \wedge x_r$ is equivalent to:

If $x_1 = 0$, return False; else if $x_2 = 0$, return False; ... else if $x_r = 0$, return False; else, return True

The number of possible decision lists is at most $4^n n!$ (choosing 0/1 for the check and return statement, and every possible ordering of the indices).

Suppose we have an algorithm LearnDL that maintains a decision list h which is consistent with all examples seen so far. Then, the following theorem applies:

Theorem 2. *With $m \geq \frac{c}{\epsilon}(n \log n + \log \frac{1}{\delta})$ examples, with probability $1-\delta$,* LearnDL *will be correct on a new example from $\mathcal{D}$ with probability $1-\epsilon$.*

Proof. The argument from Theorem 1 does not use any special properties of conjunctions or the algorithm. We can apply the same logic here.

Let $h \in H$ be a hypothesis with $Pr_{x \sim \mathcal{D}}(h(x) \neq l(x)) > \epsilon$. The probability that h is not eliminated after m examples is at most $(1-\epsilon)^m$.

The probability that any such 'bad h' survives after m examples is at most $4^n n!(1-\epsilon)^m$. (This time, substituting the size of the class of decision lists!)

Setting $4^n n!(1-\epsilon)^m < \delta$ and taking the log of both sides, we see that $m \geq O\left(\frac{1}{\epsilon}\right)(n \log n + \log \frac{1}{\delta})$ suffices. □

In general, for an algorithm that learns a consistent hypothesis from a class H, $m \geq \frac{c}{\epsilon}(\log |H| + \log \frac{1}{\delta})$ examples are sufficient to achieve this guarantee.

2 (ϵ, δ) PAC-Learning

Definition 3. *Given iid samples from an unknown distribution $\mathcal{D}$, labeled by a hypothesis ℓ from a hypothesis class H, an algorithm $\mathcal{A}$ (ϵ, δ)-PAC-learns the hypothesis class H if, with probability $1-\delta$, it produces a hypothesis $h \in H$ that correctly classifies a random sample from $\mathcal{D}$ with probability at least $1-\epsilon$, i.e.,*

$$Pr_{x \in \mathcal{D}}(h(x) = \ell(x)) \geq 1 - \epsilon.$$

PAC stands for Probably Approximately Correct. Intuitively, it states that the algorithm $\mathcal{A}$ is *probably* successful (producing a good h with probability $1-\delta$). A good h is *approximately correct* (classifying samples from $\mathcal{D}$ correctly with probability $1-\epsilon$).

Note that this definition says nothing about the efficiency of the algorithm. Usually the efficiency is judged by time or sample complexity (i.e., the time it takes to process a sample, and the number of samples that it needs).

2.1 PAC-learning Halfspaces

In section 1, we showed a heuristic for learning an arbitrary hypothesis class H. However, the number of samples needed depends on $\log |H|$. For the case of halfspaces, H is infinite, so we must use another method.

Suppose, $\|w^*\|_2 \leq 1$, $\max_x \|x\|_2 \leq 1$, and $\max_x |\langle w^*, x \rangle| \geq \gamma$. At any time, define the set of consistent hypothesis.

$$W = \{w : \|w\|_2 \leq 1, \forall i \langle w, l(x^{(i)}) x^{(i)} \rangle \geq \gamma\}$$

For the next sample $x^{(i+1)}$, divide W into two cases:

$$W_+ = \{w \in W : \langle w, x^{(i+1)} \rangle \geq 0\}$$

$$W_- = \{w \in W : \langle w, x^{(i+1)} \rangle < 0\}$$

By predicting the set with the larger volume, we guarantee that either we are correct, or the size of W is reduced by at least half.

In the next lecture, we will discuss how to use γ to bound the number of iterations we will need.