

Computational Learning Theory

Professor: Adam Klivans, Scribe: Allison Bishop

Sept. 29, 2008

1 Occam Learning for Disjunctions

We fix c in a concept class \mathcal{C} . For constants $\alpha > 0$ and $0 < \beta < 1$, we say a learning algorithm is an (α, β) -Occam algorithm if it draws m examples (labeled according to c) and outputs a hypothesis h such that h is consistent with these m examples and also satisfies:

$$\text{size}(h) \leq (n \text{ size}(c))^{\alpha} m^{\beta}.$$

We note the following theorem:

Theorem 1 (Essentially a restatement of Theorem 2.2 on p. 35 of Kearns and Vazirani.) *If \mathcal{C} is a concept class and A is an (α, β) -Occam algorithm which (for any fixed $c \in \mathcal{C}$ and any distribution on the examples) outputs a consistent hypothesis from a class \mathcal{H} after drawing $m \geq \Omega\left(\frac{\log(|\mathcal{H}|)}{\epsilon} + \frac{\log(\frac{1}{\delta})}{\epsilon}\right)$ examples (and A runs in time polynomial in n, m , and $\text{size}(c)$) then the output hypothesis of A will have error $\leq \epsilon$ with probability at least $1 - \delta$.*

We note that for A to be an efficient PAC-learning algorithm, we also need m to be bounded by some polynomial in n , $\text{size}(c)$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$.

We will apply this theorem to learning monotone disjunctions on few variables. We already know that this particular concept class \mathcal{C} is PAC-learnable, but we would like to demonstrate an algorithm for learning

$$\mathcal{C} = \{\text{monotone disjunctions of size } k \text{ over } n \text{ literals}\}$$

which only draws closer to $O(k \log n)$ examples. Intuitively, we can see that the simple elimination algorithm will not work. (The elimination algorithm starts with all the variables in the hypothesized disjunction and upon seeing an example labeled “false”, eliminates those variables which are set to “true” in this example.) For $k \log n$ examples, the elimination algorithm may output a hypothesis much bigger than $k \log n$, and so we have no guarantees on how well this hypothesis generalizes. For instance, if we are willing to output a large hypothesis, then we can just output a list of the examples we’ve seen and their labels as a hypothesis, and this will be consistent but will give us no predictive power for future examples.

Nonetheless, we will use the elimination algorithm as our starting point. We will first draw m examples and eliminate literals which are set to “true” in examples which are labeled “false.” We observe that any disjunction we form on the remaining literals will evaluate to “false” on these examples. So we only need to find a (small) disjunction on the remaining literals which evaluates to “true” on all of our examples which are labeled “true.” To describe a greedy method for forming such a disjunction, we need the following definition: a literal x_i is said to *cover* a particular example if x_i is set to “true” in this example. (Similarly, a set of literals forms a *cover* for a set of examples if each example is covered by some literal in the set.) If a set of literals forms a cover for our set of examples which evaluate to “true”, then the monotone disjunction of these literals will be consistent with all of our examples.

To greedily form a small disjunction which evaluates to “true” on all of our examples labeled “true”, we will start with an empty disjunction and iteratively choose the remaining literal x_i which covers the most of these examples to add to our disjunction until they are all covered. Now, we must bound the size of our resulting hypothesis. We let OPT represent the number of literals in the smallest cover. Since we know our concept c is a monotone disjunction of size k , it is clear that $OPT \leq k$. The number of (“true”) examples which we must cover is $\leq m$. We will assume the worst case (i.e. that $OPT = k$ and the number of examples to be covered = m).

We let U_i denote the set of examples not covered during the i^{th} iteration (so $U_0 = m$). We claim:

$$|U_{i+1}| \leq |U_i| - \frac{|U_i|}{OPT}.$$

To see why, we note that at any step i , U_i must have a cover of size $\leq OPT$, so some literal in this must cover $\geq \frac{|U_i|}{OPT}$ examples. So when we greedily choose a remaining literal which covers the most examples in U_i , we remove at least $\frac{|U_i|}{OPT}$ examples from U_i in order to form U_{i+1} . Thus, for all $i \geq 0$,

$$|U_i| \leq m \left(1 - \frac{1}{OPT}\right)^i.$$

If we set $i = OPT \log m$, we have

$$|U_i| \leq m \left(1 - \frac{1}{OPT}\right)^{OPT \log m} \leq m e^{-\log m} = 1,$$

using the inequality $1 - x \leq e^{-x}$. This shows that by choosing $i = O(OPT \log m)$ literals, we can cover all of our examples. So the size of our hypothesis h will be $O(k \log m \log n)$ bits (we need $\log n$ bits to specify each of the literals in our hypothesized disjunction). (Note that this algorithm has polynomial running time in all of the relevant parameters.)

In order for this to be an (α, β) -Occam algorithm we need:

$$O(k \log m \log n) \leq (nk \log n)^\alpha m^\beta,$$

since any concept c in our concept class can be expressed with $k \log n$ bits. This will be easily satisfied since the lefthand side grows logarithmically in m and the righthand side grows polynomially in m . To apply our theorem, we must take $m = O\left(\frac{\log(|\mathcal{H}|)}{\epsilon} + \frac{\log(\frac{1}{\delta})}{\epsilon}\right)$. Since our hypothesis is guaranteed to have size $O(k \log m \log n)$, we can fix a class \mathcal{H} of size $O(2^{k \log m \log n})$ which it always belongs to. Hence, $\log(|\mathcal{H}|) = O(k \log m \log n)$.

So we need to choose m such that

$$m = \Omega\left(\frac{k \log m \log n}{\epsilon}\right)$$

. This will hold if we choose

$$m = C \frac{1}{\epsilon} k \log n \log(k \log n)$$

for some sufficiently large constant C . To see why, note that

$$\frac{m}{\log m} = C \frac{1}{\epsilon} k \log n \left(\frac{\log(k \log n)}{\log(k \log n) - \log(\epsilon) + \log \log(k \log n)} \right)$$

for this choice of m . To incorporate the $\frac{1}{\epsilon} \log(\frac{1}{\delta})$ term in the theorem, we just add this to our m , so our final choice of m will satisfy:

$$m \geq C \left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \frac{1}{\epsilon} k \log n \log(k \log n) \right).$$

We note that this m is polynomial in terms of the desired parameters, so by theorem 1, we have obtained an efficient PAC-learning algorithm.

2 VC-Dimension

Now we will introduce the concept of VC-Dimension (Vapnik-Chervonenkis Dimension). Our motivation is the following: when a concept class \mathcal{C} is infinite, the term $\log |\mathcal{H}|$ (which will be similar to $\log |\mathcal{C}|$) in theorem 1 above is rendered unusable. Consequently, we would like to replace this with a term $VC - Dim(\mathcal{C})$ (called the VC-Dimension of \mathcal{C}) which can be small even for infinite classes ($VC - Dim(\mathcal{C})$ will be a measure of complexity for \mathcal{C} .) We suspect this will be possible since we have already seen that some infinite concept classes can be efficiently PAC-learned (e.g. axis-aligned rectangles in \mathbb{R}^2).

We let X denote an instance space (e.g. $X = \mathbb{R}^n$). We let $S \subseteq X$ be a finite subset, and \mathcal{C} be a concept class. We associate each $c \in \mathcal{C}$ with the subset of X defined by $\{x \in X | c(x) = 1\}$ (recall that c is a function from X to $\{-1, 1\}$). (Below, when we write c , we will mean this set associated with c .) We now define:

$$\pi_{\mathcal{C}}(S) = \{c \cap S | c \in \mathcal{C}\}.$$

(So $\pi_{\mathcal{C}}(S)$ is a subset of the powerset of S .)

For example, we consider the case where \mathcal{C} is the class of axis aligned rectangles in \mathbb{R}^2 and S is the following set of 3 distinct points in \mathbb{R}^2 : $(0, 0), (1, 1), (-1, 2)$. Now, we can obtain $c \cap S = \emptyset$ by taking any axis-aligned rectangle c which avoids these three points, and we can obtain $c \cap S = S$ by taking any axis-aligned rectangle which includes all 3 of these points. To get a single point of S as $c \cap S$ for some axis-aligned rectangle c , it suffices to take a small rectangle around that point which doesn't include the other two points. To get any two points of S without including the third,

we can take a rectangle with its lower lefthand corner having x -coordinate equal to the min of the x -coordinates of the two points and y -coordinate equal to the min of the y -coordinates of the two points, and its upper righthand corner having coordinates equal to the max of the coordinates of the two points (note this will exclude the 3rd point for the particular coordinates we have chosen). So in this case, $\pi_{\mathcal{C}}(S)$ is the full powerset of S (i.e. it has size $2^{|S|}$).

When $|\pi_{\mathcal{C}}(S)| = 2^{|S|}$, we say that \mathcal{C} *shatters* S . (Observe that we can also think of $|\pi_{\mathcal{C}}(S)|$ for $S = \{x_1, \dots, x_n\}$ as the number of distinct labellings $(c(x_1), \dots, c(x_n))$ induced on S by concepts $c \in \mathcal{C}$.) We define

$$VC - Dim(\mathcal{C}) = \{\text{size of the largest set that } \mathcal{C} \text{ can shatter}\}.$$

We note that $VC - Dim(\mathcal{C}) = d$ means there is some set of size d that \mathcal{C} shatters, not that \mathcal{C} shatters all sets of size d .

For the class of axis-aligned rectangles, we have $VC - Dim(\mathcal{C}) = 4$. (To see how \mathcal{C} can shatter 4 points, consider 4 points forming an axis-aligned diamond.) For the class of intervals on the real line, $VC - Dim(\mathcal{C}) = 2$ (we can shatter 2 points, but not 3). For the class of halfspaces in \mathbb{R}^2 , the VC -Dimension is 3 (note that 2 points determine a line). More generally, for the class of halfspaces in \mathbb{R}^d , the VC -Dimension is $d + 1$.